Overfitting using the MDL principle

Jakob Verbeek

Overfitting using the Minimum Description Length Principle

An investigation inspired by the paper:
"An Experimental and Theoretical Comparison of Model Selection Methods"
(Kearns, Mansour, Ng & Ron, 1997)
containing an introduction to, experiments with and analysis of:
Two-Part MDL, Predictive MDL, Guaranteed Risk Minimization
and Cross Validation

University of Amsterdam

Jakob J. Verbeek

March 8, 2000

This document is a master's thesis for the studies on Artificial Intelligence at the Computer Science department of the University of Amsterdam. Most of the experimentation and writing took place at Centrum voor Wiskunde en Informatica. The project was supervised by: Paul Vitányi, Peter Grünwald and Ronald de Wolf. This thesis will be defended at 1:30 p.m., December 16th 1998 in room P.327, Plantage Muidergracht 24, Amsterdam. The exam committee will consist of:
- Ben Kröse (UvA)
- Maarten van Someren (UvA)
- Paul Vitányi (CWI, UvA)
- Ronald de Wolf (CWI, UvA)

Universiteit van Amsterdam
Faculteit der Wiskunde, Informatica, Natuur- en Sterrenkunde
Plantage Muidergracht 24
1018 TV Amsterdam
The Netherlands

Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
The Netherlands

Abstract

This thesis investigates the experimental results obtained in the paper *An Experimental and Theoretical Comparison of Model Selection Methods* by Kearns, Mansour, Ng & Ron, published in volume 27 of Machine Learning 1997 (referred to as [KMNR97]). These 'model selection methods' are used to find a 'good' model for some observed data. The data consists of pairs (x, y). Often the 'x-data' is regarded as given and the model is a function that maps each x-value to a y-value.

We concentrate on the 'overfitting' behavior of the Two-Part Minimum Description Length (henceforth MDL) principle. Furthermore we also consider Guaranteed Risk Minimization (GRM), Predictive MDL and Cross Validation.

We verify and extend the experiments as published in [KMNR97]. The conclusions of the analysis of these experiments lead to a second series of experiments. A second series of experiments was conducted to test under which conditions the conjecture holds that removing bad models from the model class can actually decrease performance.

We conclude the following:

1. Using small model classes may lead to overfitting, if the sample is corrupted by noise.

   Using the, given, x-data to code models is one way to work with small model classes for small sample sizes.

   This phenomenon does not only affect Two-Part MDL, as suggested by the experimental results in [KMNR97], but also GRM.

   It is difficult to design a general learning algorithm that is guaranteed not to overfit the data. However, it became clear that, when we use Two-Part MDL or GRM, it is wise to keep the model class sample-independent in order to avoid overfitting.

2. Predictive MDL does overfit on the interval function selection problem. However, it is non-parametric, its performance is not dependent on the size of the model class, it does not overfit much and it overfits only for small sample sizes.

3. The MDL Two-Part principle performs well on selecting a model *class* and thus the *number* of parameters, but does not necessarily provide good estimations for the parameter *values*, so for an actual model. It is better to use Two-Part MDL just to estimate the *number* of parameters. Within the selected model class Maximum Likelihood can be used to find the best model with number of parameters suggested by Two-Part MDL.

Abstract in Dutch

In dit afstudeerproject onderzoeken we de experimentele resultaten die gepubliceerd zijn in het paper *An Experimental and Theoretical Comparison of Model Selection Methods* door Kearns, Mansour, Ng & Ron, gepubliceerd in volume 27 van Machine Learning 1997 (voortaan: [KMNR97]). Deze 'model selectie methoden' worden gebruikt om een goed model voor de geobserveerde data te vinden. De data bestaat uit paren (x, y). Vaak wordt de 'x-data' als gegeven beschouwd en is het model een functie die aan elke x-waarde een y-waarde toekent.

We concentreren op het 'overfitting' gedrag van het Two-Part Minimum Description Length (MDL) principe. Ook komen Guaranteed Risk Minimization (GRM), Predictive MDL en Cross Validation aan bod.

We herhalen de experimenten uit [KMNR97] en breiden deze uit. De conclusies uit de analyse van deze experimenten waren reden tot een tweede serie experimenten. Deze tweede serie experimenten diende om te onderzoeken onder welke condities de stelling, dat het verwijderen van slechte modellen uit de model klasse de prestaties kan verminderen, juist is.

Wij concluderen het volgende:

1. Het gebruiken van de, als gegeven beschouwde, x-data om modellen te coderen is een van de manieren om met kleine modelklassen te werken voor kleine samples.

   Het gebruik van kleine modelklassen kan leiden tot overfitting.

   Dit fenomeen doet zich niet alleen voor bij Two-Part MDL, zoals gesuggereerd werd door de experimentele resultaten uit [KMNR97], maar ook bij GRM.

   Het is een moeilijk probleem om een algemeen leer-algoritme te ontwerpen dat niet zal overfitten. Echter, het werd wel duidelijk dat het verstandig is om modelklassen te gebruiken die sample onafhanklijk zijn, wanneer we Two-Part MDL of GRM gebruiken.

2. Predictive MDL vertoont overfitting op het interval functie selectieprobleem. Voordelen van Predictive MDL zijn echter dat het non-parametrisch is, prestaties zijn niet afhankelijk van de grootte van de modelklasse, het overfit relatief weinig en de overfitting vindt alleen plaats bij relatief kleine samples.

3. Het Two-Part MDL principe is geschikt om een model*klasse* te selecteren, maar niet per definitie om een specifiek *model* te selecteren. We kunnen beter eerst Two-Part MDL gebruiken om het *aantal* parameters te schatten en vervolgens Maximum Likelihood gebruiken om het beste model te vinden, met het door Two-Part MDL voorgeschreven aantal parameters.

Acknowledgements

First of all, I would like to thank my supervisors: Peter Grünwald, Paul Vitányi and Ronald de Wolf, for reading (many) preliminary versions of the thesis and giving me many useful comments. Furthermore, I would also like to thank Petra de Wit for her comments on my use of the English language and my father for pointing out that one often needs many words to express oneself clearly. I also want to thank Peter and Ronald for the many talks we had about the thesis during lunch or work time. Many times these talks were inspiring to me. I appreciate it very much that you had so much time even when you were very busy yourselves, in particular Peter, who was at the time finishing his PhD.

This thesis served as graduation project for my studies on Artificial Intelligence, which I started in September 1994 at the WINS faculty of the University of Amsterdam. Most of the graduation project was carried out at CWI from April to July 1998. After this period a long time of rewriting and delays took place from September until November. The author of the thesis can be contacted at: `jverbeek@wins.uva.nl`

Finally I would also like to make a remark here about the names that were used for some of the methods used in the paper *An Experimental and Theoretical Comparison of Model Selection Methods* by Kearns, Mansour, Ng & Ron, as published in volume 27 of Machine Learning 1997. Some of them are prefixed with a 'K', like the MDL coding scheme that is denoted as 'KMDL'. I explicitly stress here that the 'K' is chosen purely for alphabetical reasons and no credit or discredit is suggested on the person of M. Kearns.

William of Ockham
plurality should not be assumed without necessity
(or, in modern English, keep it simple, stupid)

Contents

Chapter 1

Introduction

In this chapter we briefly describe the goals of the project, how it fits in with the field of Artificial Intelligence and what was done in this project.

1.1 Goals of the Project

In this section we introduce the subject of this thesis and we give a first motivation for the project.

Model Selection

This thesis is about Model Selection in general and the Minimum Description Length (MDL) Principle in specific. Model Selection is a central topic in statistics and in the subfield of Artificial Intelligence called *Machine Learning*. Rissanen's MDL Principle [Ris89] is one of the many known model selection methods. The idea of model selection is to model some system, called the *data generating system*, based on only a limited number of examples of the behavior of that system. The resulting model can be used to acquire knowledge of the data generating system or to predict the behavior of the system on new examples. Given m examples, the model selection problem is to select one of the many possible models that describe the data. Some models are more accurate than others or make more accurate predictions than others.

A Paper with Surprising Results

In the paper *An Experimental and Theoretical Comparison of Model Selection Methods* by Kearns, Mansour, Ng and Ron, 1997 [KMNR97] three model selection methods are compared: Rissanen's Two-Part Minimum Description Length (MDL) [Ris89], an adaption of Vapnik's Guaranteed Risk Minimization (GRM) [Vap82] and Cross Validation (CV). Furthermore some theoretical results were obtained on the class of model selection methods called 'Penalty Based Algorithms'. Both GRM and Two-Part MDL belong to this class. The algorithms in this class evaluate models using the sum of two terms. One measures the error the model makes on the data that has to be modeled. The other term is a measure of the complexity of the model. In the following we will assume that models can be ordered by their complexity. For now, think of model complexity as the number of parameters in a model. So, for instance, a 3^{rd} degree polynomial would generally be more complex than a 2^{nd} degree polynomial. Both the experimental results obtained for Two-Part MDL and the adaptation of GRM used in [KMNR97] surprised us.

Suppose that we are in a situation in which we want to model some data generating system and that we are given more and more examples of the behavior of the system. Two-Part MDL would typically start selecting simple models when the sample size is still small. As more and more examples become available Two-Part MDL usually selects more and more complex models, until it converges to some constant model complexity.[1] However, in the experiments described in [KMNR97] Two-Part MDL selected very complex models until some number m^* of examples had been reached. The model complexity of the selected models converged very rapidly if the number of examples was greater than m^*.

In addition to Two-Part MDL, we will also consider some other model selection methods. As mentioned above, GRM was adapted in [KMNR97], however it was left unclear whether or not this adaptation was justifiable in terms of the theory of Vapnik presented in [Vap82]. We will investigate whether the adaptation can be justified in terms of GRM. In [Ris89] not only Two-Part MDL is introduced but also a method closely related to it, Predictive MDL (PMDL). This model selection method is not discussed in [KMNR97]. It is similar to Cross Validation in that it evaluates a model M, which is based on a *part* of the data, by using M to model some other part of the data.

The preceding leads us to the actual goals of the project.

Goals of the Project

We formulate the following goals:

1. Is MDL applied correctly in [KMNR97]; that is, is the coding scheme used in [KMNR97] allowed by the Two-Part MDL principle?

2. Would the experimental results concerning applications of Two-Part MDL still hold when a coding scheme is used which differs from the one that was used in [KMNR97]?

3. Would the foundation of GRM, presented in [Vap82], still hold when GRM is adapted as was done in [KMNR97]?

4. How does Predictive MDL perform on the problem domain used in [KMNR97], and how is PMDL related to Cross Validation?

To answer these questions we have conducted several experiments in which we compared the performance of Two-Part MDL using several coding schemes. We also compared the behavior of Two-Part MDL with the behavior of GRM, CV and PMDL. Furthermore, we hope that with this thesis we also provide a comprehensive first introduction to Two-Part MDL, Predictive MDL and Cross Validation.

1.2 Overview of the Thesis

The thesis consists of five main parts.

1. Chapter 2 introduces to the Two-Part MDL principle.

2. Having introduced all technical preliminaries, we start discussing the first series of experiments in Chapter 3. We examine the problem domain of the first series of experiments in Section 3.1. In Section 3.2 we introduce the three different coding schemes that we used for applying Two-Part MDL. The motivation of this project can be found in more detail in Section 3.3.

[1] See [BC91].

Matters of implementing Two-Part MDL for the interval function selection problem are discussed in Appendix A.

3. The next three chapters concern our main experimental and theoretical results. In Chapter 4 we analyze, both experimentally and theoretically, the behavior of Two-Part MDL on the problem domain using the three different coding schemes. In the next two chapters we look at two other model selection methods. Guaranteed Risk Minimization is treated in Chapter 5 and in Chapter 6 we consider another form of applying the MDL principle: Predictive MDL.

4. In Chapter 7 we discuss a second series of experiments that were conducted on a different problem domain to obtain more experimental results to support our conjecture that it is good to keep the model class sample-independent in order to avoid overfitting.

5. In Chapter 8 we discuss the results obtained in the preceding chapters and draw our final conclusions from the observations and analyses.

Chapter 2

An Introduction to the MDL Principle

In this chapter we introduce the MDL principle and discuss the Two-Part MDL model selection method in full detail. We will make use of some basic notions of Coding Theory and Complexity Theory in this introduction.

2.1 Introduction to MDL

In this section we present a first and simple introduction to the MDL principle for doing model selection.

2.1.1 Model Selection

Here, we will restrict ourselves to model selection problems of the following kind:

- The data generating system is assumed to be contained in a certain restricted class of systems.
- We only consider the systems in that class as models for the data generating system.

Since the data used in the experiments is generated by systems we have implemented ourselves, the latter assumption is justified. To decide which model to pick, after observing only a relatively small number of examples of the behavior of the data generating system, is a difficult problem. To tackle this problem many model selection methods have been proposed in the history of Machine Learning.

A Simple Approach

A very straightforward approach to the Model Selection problem is to pick the model that describes the observed data, x, the best. When the model class $\mathcal{M} = \{P_1(\cdot), .., P_n(\cdot)\}$ consists of probabilistic models $P_i(\cdot)$, the best model is the model $\hat{P} = P_k$ such that $k = \arg\max_i\{P_i(x)\}$. Thus we pick the model that maximizes the probability of the occurrence of data x. This approach is known in statistics as Maximum Likelihood (ML), as it selects the model which renders the observed data most likely. The ML model selection rule is related to the MDL principle, as we will see later. However we will make clear that Two-Part MDL is a much more powerful model selection method than ML.

2.1.2 The General Idea of MDL

The general idea of the MDL principle can be explained as follows: We start by assuming there is some regularity in the observed data, because if there would be no regularity the data would be random. In this case learning and prediction would not make sense. A meaningful model that captures some of the regularity in the data, on the other hand, can be used to *compress* the data as follows: We could describe the data by first describing the regularity, the model, and next describing the deviation of the data from the described regularity. The more regularity of the data a model captures, the more it can compress the data. However, only the model and the description of the data using the model *together* give the description of the observed data. We evaluate a model using the total description length of the data using the model, so we consider the sum of the description length of the model and the description length of the data *using* the model[1]. This sum gives us an expression of the compression of the data, that is achieved by using the model. The more compression of the data some model allows, the more regularity it has captured, the better it generalizes over the observed data. We conclude that the best model is the model that compresses the observed data the most. This means we have to search for the model that yields the Minimum Description Length of the data.

In this project we consider two different forms of MDL, Two-Part MDL and Predictive MDL (PMDL). This chapter concentrates on Two-Part MDL. In the rest of this thesis, MDL refers to Two-Part MDL.

Two-Part Codes

Two-Part MDL is applied by creating a coding scheme that maps each model to a binary codeword such that from this codeword the model can be reconstructed. Each model defines, or is interpreted as, a coding scheme that is used to obtain codewords for all possible observable data. This way we describe the observations by means of a codeword that consists of two parts. First we describe a model and next we describe the data using this model. The total code length can be expressed as the sum of the code length of the model or hypothesis, H, and the description of the data, D, using the model:

$$L_C(D) = L_C(H) + L_C(D|H), \tag{2.1}$$

where $L_C(\cdot)$ is the number of bits used in the encoding using coding scheme C. The set of models that are considered is called the *model class* and it is denoted by $\mathcal{M}$.

An Example

Suppose that our observations are binary sequences. The models we use generate n repetitions of some binary sequence, s, of length l. So the model class $\mathcal{M}$ is given by:

$$\begin{aligned} \mathcal{M} &= \bigcup_{i \geq 1} \mathcal{M}i \\ \mathcal{M}_l &= \{s^n | s \text{ is a binary sequence of length } l, n \geq 1\} \end{aligned}$$

Now suppose we have observed the following string of 96 bits:

$$\begin{array}{l} 0011\ 0011\ 001\mathbf{0}\ 0011\ 0011\ \mathbf{1}011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011 \\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011\ 0011 \end{array} \tag{2.2}$$

[1] Description lengths are measured in bits.

We could model it by defining the model as one repetition of the actual string; we call this model M_A, note that $M_A \in \mathcal{M}_{96}$. Another way to model it would be by defining the model as 24 repetitions of the string '0011'; we call this model M_B. Note that M_B is not consistent with sequence 2.2. If we want to describe the original string using M_A we simply state that sequence 2.2 is given by the model. If we want to describe the sequence using M_B we have to specify that M_B gives the wrong value for the 12^{th} and the 21^{st} bit. However if we want to give a complete description of the data using *a model* we have to describe the model itself too. If we describe the data by using M_A we would need approximately 96 bits, as we just write down the sequence[2]. Using M_B we only need to describe the 4-bit sequence '0011' in approximately 4 bits, we need to specify that it is repeated 24 times and that the 12^{th} and 21^{st} bits are modeled incorrectly. This can be done by designing our descriptions in such a way that we first specify a model and next we specify the indices of the examples that are not modeled correctly by the model. It is clear that, for large strings, describing the data using the first model would cost us many more bits than describing the data using the second model. Two-Part MDL would therefore prefer M_B over M_A.

Summarizing

MDL advocates selecting the model that achieves the greatest data compression obtainable using a model from the model class. We saw before that, in general, a model allows a certain amount of compression of the data if it captures some of the regularities present in the data. However, when a model is used which can only be described using many bits, for example an exact description of the data, the rate of compression that was obtained by the very short description of the data is reduced by the long description of the model. This trade-off between model complexity and how well the model fits the data is represented by the addition of the two terms in Equation 2.1.

With high probability the regularity in the data is induced by the data generating function and not by the noise, for large samples. So the regularity in the data can be seen as a representation of the data generating function. By modeling the observed data we hope to come up with good models for the data generating function. The measure of goodness of a model in terms of data compression seems to be a reasonable one, if we want to model some data generating function. See [Grü99] for a more elaborate discussion of the MDL principle, [Ris89] gives full details.

2.2 Codes and Probabilities

In this section we discuss coding schemes and the connection between coding schemes and probability distributions. We start with coding schemes.

2.2.1 Coding Schemes

How do we encode models and data using models? We do so by using some fixed *coding scheme*. There is some *source* domain, X of which we want to describe some objects. The coding scheme assigns a *codeword* to each object in X. A codeword is a string of symbols from some alphabet D. From the codeword, we want to be able to reconstruct, the source object(s) again.

Now we make things more formal and impose some restrictions on the coding schemes we want to use.

[2]We will explain later why exactly 96 bits are not enough.

Definition 2.1 *A* **coding scheme** C *is a mapping that maps objects from some set* X *to objects in* D^*.

In this thesis we use $D = \{0, 1\}$ and assume that C assigns a unique element of D^* to each $x \in X$. We denote the *codeword* that coding scheme C assigns to some object $x \in X$ by $C(x)$. There are several restrictions on the coding schemes that we will use to code models and data with respect to some model. The first of these restrictions is that the coding scheme should be non-singular.

Definition 2.2 *A coding scheme* C *is called* **non-singular** *if* C *assigns a different codeword to every object* $x \in X$*:*

$$x_i \neq x_j \Rightarrow C(x_i) \neq C(x_j) \tag{2.3}$$

If the coding scheme is non-singular, the codeword $C(x)$ is an unambiguous description of $x \in X$.

With a slight abuse of notation, we define the codeword for a sequence of objects from X as the concatenation of the codewords of those objects:

$$C(x_1 x_2 ... x_n) = C(x_1)C(x_2)...C(x_n) \tag{2.4}$$

The next restriction is that we demand the coding schemes to be uniquely decodable.

Definition 2.3 *We call a coding scheme* **uniquely de-codable** *if for every two sequences of objects from* X *it holds that if they are different then their codewords will also be different.*

The coding scheme is non-singular for sequences of objects if it is uniquely decodable. The last restriction assures that each sequence of concatenated codewords has only one possible sequence of source objects. However, we can still have a situation in which we have to look ahead in the string of concatenated codewords to be sure that we have reached the end of some codeword or that we are still in the middle of some other codeword that starts with the same binary sequence.

Example 2.2.1 *Let* C *be a coding scheme such that* $X = \{A, B\}, D = \{0, 1\}$ *and* $C(A) = 10, C(B) = 1$. *This coding scheme is uniquely de-codable, as we can see that whenever a '1' is followed by another '1' then the first '1' denoted 'B'. The string '111' is easily decoded as 'BBB'. However, if we decode the string '101011' then we cannot decide without looking at the second bit, whether the first bit belongs to the codeword of 'A' or to the codeword of 'B'.*

This motivates another restriction on the coding schemes.

Definition 2.4 *A coding scheme is* **prefix-free** *if no codeword is the prefix of any other codeword.*

It is easy to see that every prefix-free coding scheme is uniquely de-codable. It turns out that for every uniquely de-codable coding scheme C_{ud} there is a prefix-free coding scheme C_{pf} which assigns to all $x \in X$ codewords of the same length as C_{ud}.

Example 2.2.2 *Consider the coding scheme* C *introduced in Example 2.2.1. By a slight modification we can turn* C *into a prefix-free coding scheme* C'. *Simply take* $C'(A) = 01$ *and* $C'(B) = 1$. *If the first symbol of a codeword is a '1' we know that a 'B' was encoded. Furthermore we know that a new encoding starts at the second symbol in the codeword. If the first symbol would have been a '0' we would know that a 'A' was encoded and that a new encoding starts at the third symbol in the codeword.*

Hence it suffices to restrict attention to prefix-free coding schemes if we want only to consider uniquely de-codable coding schemes, this is proven in [BC91] Section 5.5. If a coding scheme is prefix-free we can identify the end of each codeword without looking ahead[3]. When applying MDL we restrict ourselves to using prefix-free codes, we return to this issue in the next section.

2.2.2 From Codes to Probabilities

In this subsection we discuss one of the relations between coding schemes and probability distributions.

It is a basic fact of information theory, see [CT91], that for each prefix-free coding scheme, C, that assigns a codeword of length $|C(x)|$ to each x in a finite or countably infinite set of events X, the **Kraft Inequality** holds:

$$\sum_{x \in X} 2^{-|C(x)|} \leq 1 \tag{2.5}$$

Conversely, if a set of codeword lengths satisfies the Kraft Inequality, then there is a prefix-free coding scheme that yields these codeword lengths.

An interesting question when designing prefix-free coding schemes is which coding scheme would yield the minimal expected code length? Shannon has shown that if the objects $x \in X$ have a probability of occurrence of $p(x)$ and the code lengths are chosen to be $-\log p(x)$, then the expected average code length is minimized[4]. It is easy to check that these codeword lengths satisfy the Kraft Inequality:

$$\sum_{x \in X} 2^{-|x|} = \sum_{x \in X} 2^{-(-\log p(x))} = \sum_{x \in X} p(x) = 1$$

In practice $-\log p(x)$ is in many cases not an integer and we choose codeword lengths of $\lceil -\log p(x) \rceil$, the smallest integer not smaller than $-\log p(x)$. As these codeword lengths are only longer the Kraft Inequality still holds and the codeword lengths are less than one bit longer than $-\log p(x)$. This single bit will not be significant in practice. Therefore we will ignore the fact that codeword lengths should be integer-valued.

The famous Shannon-Fano coding scheme assigns codewords of length $-\log p(x)$. There is a simple way to generate codewords for elements of the set X which have these lengths, see [CT91]. When applying the MDL principle we are not interested in the actual codes but just in the code lengths, as these code lengths give us the values we want to minimize. Therefore we will not discuss the way to generate the codewords for a Shannon-Fano coding scheme here.

The Kraft Inequality gives us an important connection between coding schemes and probability distributions. Using the Kraft Inequality we can let coding schemes define probability distributions and vice versa[5]. We let coding scheme $C(\cdot)$ define probability distribution $P_C(\cdot)$ over the source domain X by letting

$$\forall x \in X : P_C(x) = 2^{-|C(x)|},$$

[3] Another way to make sure we always identify the end of each codeword would be to place a comma between each codeword to separate them. But in that case we would need an extra symbol, the comma, to do so and the alphabet D is not binary anymore in such a situation. As the theory of MDL as we discuss it uses codewords that are strings of zeros and ones we do not want to use the extra symbol. In fact, using the extra symbol would not be very efficient as we would reserve it only to denote comma's, so the extra symbol will not be used much.

[4] Whenever we use $\log(\cdot)$ throughout this thesis we refer to the logarithm with base 2.

[5] Normalization is needed in case of non-integer code lengths and in cases where the sum of Equation 2.5 is strictly smaller than 1.

where $|C(x)|$ denotes the length of $C(x)$ in bits. We let probability distribution $P(\cdot)$ define some coding scheme $C_P(\cdot)$ over the domain X of P by letting

$$\forall x \in X : |C_P(x)| = -\log P(x).$$

So we let $P(\cdot)$ define a set of codeword lengths rather than an actual coding scheme. As the codeword lengths are all we need when applying Two-Part MDL this forms no limitation.

In the next two sections we will discuss the principles of an *ideal* version of MDL.

2.3 Two-Part Codes and Kolmogorov Complexity

In this section we introduce Prefix Kolmogorov Complexity to introduce an *Ideal* version of Two-Part MDL.

2.3.1 Informal Introduction

By evaluating hypotheses by the Two-Part description length of the data that they yield, there is a trade-off between how precise a model describes the observations and the description length of that model, as is illustrated by Equation 2.1. According to the MDL principle [BRY97], Section 1, "the problems of modeling and inference are not to estimate the true data generating functions, but rather to search for good models of the data where the goodness is defined in terms of code length", so to look for the model or class of models that best capture the regularity of the data.

The results of the theory of algorithmic complexity, developed by Solomonoff [Sol64], Kolmogorov [Kol65] and Chaitin, can be applied to arrive at an *ideal* version of MDL.

Roughly speaking the Kolmogorov Complexity of a string x is the length of the shortest computer program that prints x and then halts. Here the program should be written in some fixed universal language, such as some specific computer programming language. One may identify a model for the data with a computer program that *generates* the data, i.e. prints the data and then halts.

One of the main results of the theory of algorithmic complexity is that there can be no algorithm that computes this algorithmic complexity. As a consequence of this result there can also be no algorithm which, for arbitrary data D, gives us the shortest program that prints D and then halts. Thus if we use the model class of all computable models, i.e. using all models that can be identified with some computer program, there is also no algorithm that gives the best model, in terms of minimum description length, for some observed data. So, in practice, we will always have to work with approximations of the algorithmic complexity and some restricted model class. Below we define a version of MDL that is directly based on Kolmogorov Complexity. It inherits the non-computability of Kolmogorov Complexity, hence it can not be used in practice. Therefore we call it *Ideal* MDL.

We will first define Prefix Kolmogorov Complexity informally. Then we will give a more formal definition. The Prefix Kolmogorov Complexity, or algorithmic prefix complexity, $K(x)$, of a string x is the length of the shortest computer program for a universal Prefix Turing Machine that outputs x and then halts.

The Universal Prefix Turing Machine is the equivalent of a Universal Turing Machine when using prefix machines. A Prefix Turing Machine is a Turing Machine for which the set of programs is a prefix-free code. So no program is a prefix of another program.

The Kolmogorov complexity is a universal assignment, so independent of the Universal Turing Machine or programming language at hand, up to a constant. This

constant is independent of the actual string x, but is needed for translation of the program between different Universal Turing Machines or programming languages. See [LV97] for more details on Kolmogorov Complexity.

2.3.2 Making Things Formal

We start by formally defining the Prefix Kolmogorov Complexity.

Definition 2.5 *Let U be some fixed Universal Prefix Turing Machine. Let $x \in \{0,1\}^*$. Let $\mathcal{P}$ be the set of programs (that is: inputs) for U. Let $U(p)$ be the output of U for program p. $K(x)$, the* **Prefix Kolmogorov Complexity** *of x, is defined as:*

$$K(x) = \min\{|p| : p \in \mathcal{P}, U(p) = x\}$$

Now we give the **Invariance Theorem** that was informally introduced above:

Theorem 2.1 *Let U and T be two different Universal Prefix Turing Machines. Let $x \in \{0,1\}^*$. Let $K_M(x)$ be the Prefix Kolmogorov Complexity of x on machine M, then*[6]

$$K_U(x) = K_T(x) \pm c(U,T)$$

where $c(U,T)$ is independent of x, but dependent on U and T.

Now we give a simple example of Kolmogorov Complexity to clarify the concept.

Example 2.3.1 *Let x be some binary string. We have that $K(x) \leq 2|x| + c$, as for each string, x, we can write a program that looks like "write x". This will require some bits for the "write" statement, say c bits. Next, we need some bits for a prefix-free version of x. We could decide to make a prefix-free code for binary strings by inserting a zero after each bit, except for the last, and inserting a one after the last bit of the string. This would take us $2|x|$ bits.*

See Definition 2.7 for a better bound on K(x).

Models as Programs

In the Kolmogorov Complexity framework, the shortest program for the data serves as the optimal model. Consider the observed data as a binary string, we denote this string by D. Next we identify models with programs. The Two-Part code becomes the concatenation of two programs in this context. The first program is the model, which outputs some binary string. The second program acts upon the output of the first program in such a way that the output of the second program is *exactly* the string D.

We say the first program describes the *regular* or 'interesting' part of the data, as the model tells us something about the structure of the data. The second program is said to describe the *random* or 'uninteresting' part of the data. The random part of the data is said to be uninteresting as it just tells us something about the details of the actual observed data, the exceptions of the data on the modeled regularity, and is unlikely to contain information about future observations. Now the Two-Part MDL principle would advocate a choice for the model which yields the shortest program, which is a concatenation of two programs, for the data.

[6] See [LV97] for a proof of the Invariance Theorem.

Why use Prefix-free Coding Schemes?

Suppose we want to know how likely it is that some Turing Machine outputs a certain finite binary sequence D and then halts. We want to use as input some random binary sequence P of some length m. If we use a normal Universal Turing Machine a problem arises, because only for Universal Prefix Turing Machines we have that no program is a prefix of another program. Using a normal Universal Turing Machine one would also need to mark the end of the program (input) with a special symbol, usually denoted by '#'. So only using a Universal Prefix Turing Machine we can use some purely binary input.

Now recall the relation between coding schemes and probability distributions introduced in Section 2.2.2 on page 13. This relation was possible using the fact that we use prefix-free codes. Using the relation between coding schemes and probability distributions we can reinterpret Ideal Two-Part MDL as picking the computer program that is most likely to have generated the data D, as we will see in the next section.

2.3.3 Kolmogorov Minimal Sufficient Statistic

Now we introduce a concept closely related to the Two-Part MDL coding scheme. For each string x there are a number of programs that reach $K(x)$ within a given constant c, their length is $K(x) + c$. Among those programs there are programs that describe x in two parts. First some property A of x is optimally described in $K(A)$ bits, the Kolmogorov Complexity of the codeword for A using some coding scheme. The description of property A specifies that the described string is in the set of strings that satisfy property A, we will also call this set A. Next, we describe the index of x in this set A in $\log |A|$ bits, where $|A|$ is the number of elements in the set A. Let A_k^* be the smallest set containing x that can be described in at most k bits. The description of x using A_k^* would take at most $k + \log |A_k^*|$ bits. Now we define the Kolmogorov Minimal Sufficient Statistic[7].

Definition 2.6 *Let k^* be the least k such that*

$$k + \log |A_k^*| \leq K(x) + c \tag{2.6}$$

Here A_k^* is again the smallest set containing x that can be described in at most k bits, so $K(A_k^*) \leq k$, and c is some small constant.

We call the property A a **Kolmogorov Minimal Sufficient Statistic** *if A is the smallest set containing x that can be described in at most k^* bits.*

We could describe a string D using a Kolmogorov Minimal Sufficient Statistic. First we describe some set A in $K(A)$ bits and next we describe D by giving its index in A in $\log |A|$ bits. Such a description looks very much like a description using a MDL Two-Part code. We see that in the description of a string using the minimal sufficient statistic, the lefthand side of Inequality 2.6, there is also some trade-off as in Equation 2.1, the description length of the MDL Two-Part codes. However, the models that are used when applying MDL differ from the sets involved with the Kolmogorov Minimal Sufficient Statistic in several ways: We do not require the first part of the Two-Part code to be the description of a set which contains x, but let it be a description of a (statistical) model for the observed data. We therefore replace the $\log |A|$ term with the number of bits that is needed to describe x given the model. Furthermore we make two concessions. First, we do not require the length of a description of a model M to be equal to $K(M)$, the length of the shortest computer program that generates M. We do so because this

[7] See also [CT91] page 176 and 182.

would computationally not be realizable. Second, we do not search for the optimal model among all computational models, rather we restrict the search to take place within a given set of models.

In the following section we link the MDL principle more formally to the Kolmogorov complexity.

2.4 MDL and Bayes' Rule

In this section we discuss the link between MDL and Bayes' Rule. We start by giving a derivation of MDL from Bayes' Rule. In the second subsection we consider under which conditions the derivation holds.

2.4.1 Derivation of MDL from Bayes' Rule

In this section we derive MDL from Bayes' rule. Using Bayes' Rule we obtain an intuitive foundation for MDL[8].

MDL makes certain assumptions about the relation between the observed data and the selected hypothesis[9]. We consider them in the next subsection. First of all, to get a better idea of the foundation of MDL, we link an ideal version of Two-Part MDL to Bayes' Rule and then move to a practically applicable version of the MDL principle. The Bayesian framework assumes there is some 'prior probability' on the hypothesis space. In deriving MDL from Bayes' Rule we use a very natural and intuitive distribution as prior. Bayes' Rule is given by:

$$Pr(H|D) = \frac{Pr(D|H)P(H)}{Pr(D)}. \tag{2.7}$$

In (2.7) $P(H)$ is the prior probability of some model or hypothesis $H \in \mathcal{M}$, $\mathcal{M}$ is the model class under consideration, D is the set of observed data and

$$Pr(D) = \sum_{H' \in \mathcal{M}} Pr(D|H')P(H') \tag{2.8}$$

In this probabilistic setting, the task of inductive inference is to find the model H that maximizes $Pr(H|D)$, the probability of the hypothesis given the observed data, and thus maximizes (2.7) for given D and P. Taking the negative logarithm on both sides of Equation 2.7 we should minimize $-\log Pr(H|D)$ over H, or equivalently minimize:

$$-\log Pr(D|H) - \log P(H) + \log Pr(D). \tag{2.9}$$

Because $Pr(D)$ does not depend on H we can discard this term in the minimization process. Let us now substitute (we will discuss whether such a substitution is allowed in the next section) the universal distribution[10] $\mathbf{m}(\cdot)$ for $Pr(\cdot|\cdot)$ and $P(\cdot)$, i.e. we set

$$-\log Pr(D|H) - \log P(H) = -\log \mathbf{m}(D|H) - \log \mathbf{m}(H) + \mathcal{O}(1). \tag{2.10}$$

Theorem 4.3.3 of [LV97] says that (under conditions we will consider in the next subsection) $\log \mathbf{m}(x) = -K(x) \pm \mathcal{O}(1)$. If we also make this substitution we should minimize

$$K(D|H) + K(H) \tag{2.11}$$

[8] It is at least intuitive for Bayesians.

[9] We have taken these assumptions from [LV97] Section 5.5 on Hypothesis Identification by Minimum Description Length. It gives a clear and extensive explanation of the MDL principle.

[10] See [LV97], Section 4.3.

over H. The minimization of the quantity given in (2.11) gives us the ideal and absolute form of MDL. The first term gives the code length for a description of the observations using the model. The second term gives the number of bits needed to code the model.

2.4.2 From Ideal MDL to Practical MDL

As the Kolmogorov complexity, $K(x)$ of an object x is not computable, we cannot use MDL in its purest form. We are thus forced to use prefix-free coding schemes that perform sub-optimally, that is to say, yield code lengths that differ from the Kolmogorov Complexity. The question is which computable prefix-free coding schemes are allowed and/or are the best. A second point that needs our attention is the issue of exactly when are we allowed to make the substitution that gave us Equation 2.10 and led to (2.11).

To start with the last question; there are two conditions for using the universal distribution $\mathbf{m}(\cdot)$. The first is that H_{mdl}, the hypothesis selected by the MDL principle, is P-random for the prior distribution P. This is the case when:

$$-\log P(H_{mdl}) \leq K(H_{mdl}) + c,$$

where c is some constant independent of H_{mdl} which determines the degree of P-randomness[11]. So, in general, we should make sure that our coding scheme will give a very short description of H_{mdl} that leaves as little regularity in the code as possible. The second condition is that the data, D, is $Pr(\cdot|H)$ random, so we should have that:

$$-\log Pr(D|H) \leq K(D|H) + c,$$

where c is some constant independent of H which determines the degree of $Pr(\cdot|H)$-randomness. So for coding the data with the model, we should also use a coding scheme that yields very short codes, that is a coding scheme that leaves as little regularity as possible in the code.

We see that the second question reduces to the first, that is the substitution of $\mathbf{m}(\cdot)$ is allowed only if we use code schemes which yield code lengths that approximate $K(\cdot)$ fairly well for the data at hand.

2.5 Coding Data and Models

In this section we consider how we can code some data D using a model from a model class $\mathcal{M}$. First we introduce the sender receiver model.

The Sender-Receiver Model

MDL is often presented in the sender-receiver model, in which A(lice) wants to send B(ob) some information or data, D. Two-Part MDL can be seen as follows in this context. Alice first sends the description of some model, M, to Bob. Next she sends a description of the data, D, to him. This description is done with the help of M. Suppose Alice wants to minimize the length of the communication. She has to pick the model that minimizes the sum of the description length of the model and the description length of the data using the model. The sender-receiver model often offers a clarifying view on the MDL principle, as it makes clear what we have to code and in which way.

[11] For more details on randomness see Section 4.3 of [LV97].

Coding the Model

First we consider how we can code some data using a model class. After that we reconsider the example given in Section 2.1.2 on page 10 in more detail.

Suppose we want to code some data using the Two-Part code. First, we have to code the model in a prefix-free fashion. As in general we have no prior distribution over the model class, we cannot use the Shannon codeword lengths $-\log p(\cdot)$. So we have to construct a prefix-free coding scheme that yields codeword lengths which approximate as well as possible the Kolmogorov Complexity of the models. We can encode parametric models by first encoding the number of parameters and then the actual set of parameter values in a prefix-free way. An easy, but inefficient way to make codes prefix-free is by inserting a zero after each symbol in the original codeword and end the new codeword with a '1'. If we decode the new codeword, we know that after reading a 1 on an even position in the code we have reached its end. This method would assign to each codeword x a prefix-free codeword of length $2|x|$. Another, more efficient, way to turn a coding scheme C into a prefix-free coding scheme C' is first to describe the length of the codeword in a prefix-free way, using $2\lceil \log |x| \rceil$ bits as the length of x in bits can be described in $\lceil \log |x| \rceil$ bits, and then literally give the codeword[12]. The only constraint on C to turn it into a prefix-free coding scheme this way is that C should be non-singular. The latter method will only cost us $|x| + 2\lceil \log |x| \rceil$ bits instead of $2|x|$ bits, with $|x|$ the length of the codeword x in bits. Of course this trick can be repeated by first describing *the length of the length* of x in $2\lceil \log \lceil \log |x| \rceil \rceil$ bits, then describe the length of x in $\lceil \log |x| \rceil$ bits and finally describe x in $|x|$ bits. However, the bits gained by repeating the trick will get fewer and fewer.

Note that if we encode integers in the standard way, i.e. we use the coding scheme:

x	$C(x)$	x	$C(x)$	x	$C(x)$
1	0	5	100	9	1000
2	1	6	110	10	1001
3	10	7	101	11	1010
4	11	8	111	...	...

we have that $|x|$ is given by $\lceil \log x \rceil$. We define $L^*(n)$ as the code length of an integer n, coded using the trick defined above. In this thesis we will satisfy ourselves with applying the trick one time.

Definition 2.7 *Let $L^* : \mathbb{N}^+ \to \mathbb{N}^+$ be a function that assigns to each integer the number of bits needed to encode that integer in a prefix-free fashion.*

$$L^*(i) = \begin{cases} 3 & i \leq 2 \\ \lceil \log i \rceil + 2\lceil \log \lceil \log i \rceil \rceil & i > 2 \end{cases}$$

So the code length $L^*(\cdot)$ assigns to each integer i is based on encoding i and then encoding the length of i in a prefix-free fashion.

When using a coding scheme as just described we can code models of arbitrary complexity (i.e. number of parameters). We can, for example, make codes for all polynomials that will describe the degree of the polynomial as well as its actual parameter values. One usually codes a model as follows: One first encodes, in a prefix-free fashion, the number of parameters, t, then the number of bits, p, with which parameters will be encoded, the *precision*. When decoding we determine the number of bits that will follow after we have decoded p and t, namely pt. So the last

[12] In this way a uniquely de-codable code is constructed, because we know that the codeword ends after $|x|$ bits if we have read the first part of the codeword, which describes $|x|$ in $\lceil \log |x| \rceil$ bits. Remember that for every uniquely de-codable coding scheme, there is a prefix-free coding scheme which yields the same codeword lengths.

part of the encoding of the model, the actual parameter values, does not have to be encoded in a prefix-free fashion. The encoding of p and t can be seen as encoding the length of the code word for the parameter values.

Coding the Data

After we have coded the model in a prefix-free way, we can code the data using a coding scheme that was constructed using this model. If we use probabilistic models we can use the Shannon-Fano coding scheme. Remember that this coding scheme assigns to each object, x, a codeword length of $-\log p(x)$. The more likely the model renders some data sequences, the shorter the code for that data sequence is. If the model captures the regularity in the data fairly well, then the data will have high probability and the code for the data using the model will be short. This is a desirable feature of coding schemes. We will continue to look at the construction of codes for models and data in Section 3.2. Now we reconsider the example given in Section 2.1.2 on page 10.

Example 2.5.1 *Recall that our observations consisted of binary sequences and that our models are defined as: 'n repetitions of string s'. Let l denote the length of s. Let $L_M(D) = L(M) + L(D|M)$ denote the total code length that is assigned to data D if we code it using model M. Let $L(M)$ denote the code length of model M and let $L(D|M)$ denote the code length of D given M.*

Now we can code a model M by giving a prefix-free description of s and a prefix-free description of n. We have seen that we can do so in $L^*(l) + l + L^*(n)$, so $L(M) = L^*(l) + l + L^*(n)$. First we describe the length of s in a prefix-free fashion in $L^*(l)$ bits. Next we describe s in l bits and finally we have to describe n in $L^*(n)$ bits. So we would describe M_A in $L(M_A) = L^*(96) + 96 + L^*(1) = 112$ bits. For a description of M_B we would need $L(M_B)L^*(4) + 4 + L^*(24) = 19$ bits. For M_B we also have to code the indexes of the two bits that it does not predict correctly. We give those indexes in $L^*(12)$ and $L^*(21)$ bits respectively. So $L(D|M_B) = L^*(12) + L^*(21) = 19$.

Now we arrive at $L_{M_A}(D) = L(M_A) = 112$ and $L_{M_B}(D) = L(M_B) + L(D|M_B) = 38$.

2.6 Summary

Now we will briefly review the main points of the last chapter.

1. MDL presents Inductive Inference in the light of **data compression**. The MDL principle will prefers the model that allows maximal compression of the observed data.

2. When applying Two-Part MDL we will restrict ourselves to using **prefix-free coding schemes**. Only prefix-free codes give a fully self-contained description as the code length is contained in the code.

3. If we use prefix-free coding schemes we can **identify probability distributions and coding schemes**. The Kraft Inequality allows us to link probability distributions and coding schemes. This makes it very easy, if we use probability distributions as models, to compute code lengths for some data given a model.

4. Two-Part MDL can only be applied if we use an **approximation of the ideal version of MDL**. Due to the non-computability of the Kolmogorov Complexity we can not determine it for models nor for some data relative to some model.

Chapter 3

Experiments

In this chapter we look in more detail at the various ways we applied the Two-Part MDL principle on the interval function selection problem. This is the problem domain used in the article [KMNR97]; it will be the problem domain used in the first series of experiments. In the first section, we introduce the problem domain. The second section deals with the question of how we can construct codes for models and codes for the data using a model. In the third section, we motivate the goals of the project using the different coding schemes that were presented in Section 3.2. In Appendix A issues of implementation are discussed. In this appendix we consider how we can obtain good models for the data without having to search the total model space and present the algorithms that were implemented. We advise the reader who wants all detail to read appendix A after this chapter.

3.1 The Problem Domain of the Experiments

The interval function selection problem, as defined in [KMNR97], can be defined as follows. The data source is a function $f_d(x)$ that divides the interval $[0,1]$ into $d+1$ equal intervals, each of length $\frac{1}{d+1}$(see Figure 3.1). The examples are drawn according to a uniform distribution on $X = [0,1]$. The data sequences consist of pairs (x,y) such that $x \in X$ and $y \in Y = \{0,1\}$.

The value $f_d(x)$ is zero in the first and all other odd intervals, it is 1 in all other intervals. So the function value switches value d times on the interval $[0,1]$. We call the points where an interval ends a *switch*, as $f_d(x)$ switches value on those points. We define the model class $\mathcal{M}_k$ as the set of all functions $h_k : X \to Y$ that switch value k times on $[0,1]$ and the function value is zero in the first interval. All $h_k \in \mathcal{M}_k$ can be completely described by specifying the k values where it switches

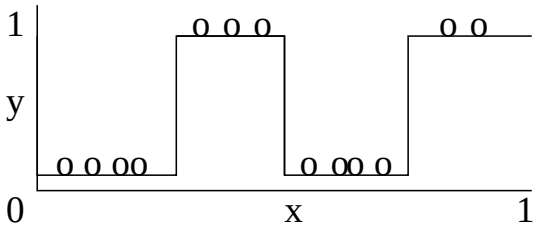

Figure 3.1: Plot of a sample. Each 'o' represents a single example. The line represents the data generating function. All examples are labeled according to the data generating function.

value, so the functions in $\mathcal{M}_k$ have k parameters. We let

$$\mathcal{M} = \bigcup_{0 \leq k \leq m} \mathcal{M}_k,$$

with

$$\mathcal{M}_k = \{f : [0,1] \rightarrow \{0,1\} | f \text{ switches value } k \text{ times}\}$$

We refer to the functions in $\mathcal{M}$ in terms of models or hypotheses, as the functions model the data up to some degree and each model can be seen as a hypothetical statement that the data is best modeled by that model. Note that the data source $f_d(\cdot)$ is in the model class $\mathcal{M}$. We will sometimes use the set of locations of the switches in $h_t(\cdot)$ to denote $h_t(\cdot)$. Hence, when we speak about the 'hypothesis' $\{x_1, x_2, .., x_t\}$, we mean the function h_t that has a switch at x_1, x_2, etc.

The task of the inference algorithms discussed in this thesis is to output a function $h_t \in \mathcal{M}_t \subset \mathcal{M}$ that has an error $\epsilon(h_t)$ as small as possible. Now we give some definitions to make things formal:

Definition 3.1 *A* **sample** $S(m)$ *is a sequence of* m *pairs* (x_i, y_i) *such that* $x_i \in [0,1]$, $y_i \in \{0,1\}$ *and the* x_i *are ordered from small to large. We call the pairs* (x_i, y_i) *that are part of the data* **examples**. *We call* (x_i, y_i) *positive if* $y_i = 1$, *otherwise we call it negative.*

Definition 3.2 *The* **error** *or* **generalization error**, $\epsilon(h_t)$, *of a hypothesis* h_t *relative to some function* $f_d(\cdot)$, *is given by:*

$$\epsilon(h_t) = Pr_{x \in X}\{h_t(x) \neq f_d(x)\},$$

where the probability is taken with respect to the uniform distribution over the example space X, *in our case: [0,1].*

Definition 3.3 *The* **training error,** $\hat{\epsilon}(h_t, S)$, *of a hypothesis* h_t *on some sample* $S = S(m) = ((x_1, y_1), ..., (x_m, y_m))$, *is given by:*

$$\hat{\epsilon}(h_t, S) = \frac{|\{x_i : h_t(x_i) \neq y_i, 1 \leq i \leq m\}|}{m},$$

often $\hat{\epsilon}(h_t, S)$ *is denoted as* $\hat{\epsilon}(h_t)$, *if* S *is clear from the context.*

Ideally, the learning algorithm should output the model that models the data generating function the best. This is the model that makes the smallest generalization error on the data generating function. However, the training data is corrupted with some noise to make the setting more interesting and more realistic. There may be some very complex model that fits the data exactly, the training error is zero. However we have with overwhelming probability that the best model will not have zero training error, as there is noise present in the sample. Therefore we cannot rely on the training error to find the model that yields the lowest generalization error.

3.2 Coding Models and Data

In this section we will first describe a method to code some data sequence given some model that is an indicator function. In the second subsection we discuss three methods to encode models for interval functions.

3.2.1 Coding Data Given some Indicator Function

First we define indicator functions:

Definition 3.4 *A function f is an* **indicator function** *if $f : X \to \{0, 1\}$.*

An indicator function f can be used to indicate whether some object $x \in X$ belongs to some concept defined on X. If x belongs to the concept then $f(x) = 1$, otherwise $f(x) = 0$. Therefore, model selection problems involving indicator functions are also known as Concept Learning.

If M is an indicator function, then we can describe a given data sequence

$$S(m) = (x_1, y_1), (x_2, y_2), ..., (x_m, y_m), x_i \in X, y_i \in Y \tag{3.1}$$

with a given model M as follows:

First describe the sequence of m x's, since we need to model the dependence $X \to Y$. If we feed the model, M, a sequence of m x-values, it outputs a sequence from $\{0, 1\}^m$. Second, to code the values $y_1, y_2, \ldots, y_m$, all we need to do is code the errors of the model. We can do this again by giving a sequence of m zeros and ones. Each 1 in the latter sequence tells us where the value predicted by the model (so the value in the first sequence) is incorrect. Now let F_0 and F_1 be the frequency of zeros and the frequency of ones in the third sequence respectively. We take the code lengths $L(\cdot)$ as prescribed by Shannon[1] to get the minimal expected total code length: $L(0) = -\log F_0 = -\log(1 - \hat{\epsilon})$ and $L(1) = -\log F_1 = -\log \hat{\epsilon}$, here $\hat{\epsilon}$ is short for $\hat{\epsilon}(M, S)$, defined as in definition 3.3. The number of ones that has to be coded is $m\hat{\epsilon}$, the number of zeros is $m(1 - \hat{\epsilon})$. Let

$$\mathcal{H}(x) = -(x \log x + (1 - x) \log(1 - x))$$

denote the entropy function for a binary alphabet; see for example [CT91]. So the total code length of the second sequence is:

$$-m(\hat{\epsilon} \log \hat{\epsilon} + (1 - \hat{\epsilon}) \log(1 - \hat{\epsilon})) = m\mathcal{H}(\hat{\epsilon}).$$

We can describe the observed data given the model in $m(\mathcal{H}(\hat{\epsilon}) + p)$ bits, where p is the number of bits needed to code one x value. Usually, the term mp for the x data in the code length of the data given some model is ignored in the minimization process, as it is assumed that the x sequence will have the same description length for all models or is simply given.

3.2.2 Coding Interval Functions

In this subsection we discuss several ways to construct codes for interval functions, the model class we use in our first series of experiments.

When making codes according to the MDL principle, assuming that the x-data is not present, we are forced to code the parameters of our model independent of the data. In the process of Model Selection we need to optimize the precision used to code the parameters of the model. In a sophisticated version of MDL this optimization does not require equal precision for each parameter. But for reasons of simplicity we code each parameter using an equal number of bits.

In the experiments we have used three different coding schemes for the MDL model selection method. We discuss them below.

[1] See [CT91] for why this leads to the minimal expected code length.

$\mathbf{MDL_{tp}}$

If we take p to be the optimal[2] number of bits to describe a parameter value and t to be the number of parameters, then we can describe a model using $L^*(t) + L^*(p) + tp$ bits. We first encode the parameter precision and the number of parameters in a prefix-free fashion. Once the decoder has these values, he can compute how many bits will follow for the actual description of the parameter values, namely tp bits. We call MDL using this coding scheme MDL_{tp}, as it uses roughly tp bits for a t-parameter model. In fact there is also one bit needed to specify whether the first interval is labeled positive or negative. As this bit would be needed in each hypothesis it does not influence the minimization process, we therefore ignore it.

$\mathbf{MDL_{pss}}$

The codes used in the MDL_{tp} method are not very efficient; the same information can be expressed in fewer bits.

We call a coding scheme $C : X \to \{0,1\}^*$ *redundant* if there is a coding scheme $C' : X \to \{0,1\}$ such that for all $x \in X : |C(x)| \geq |C'(x)|$ and for at least one $x \in X : |C(x)| > |C'(x)|$.

Below we show that MDL_{tp} is redundant. To squeeze (some of) the redundant bits out we construct a code for a hypothesis $h_{t,p}$ such that we: First, again, code the number of bits p, that was used to code one parameter value, in $L^*(p)$ bits. Second, code the number of parameters t of the hypothesis in $L^*(t)$ bits. Third, code the index number of the hypothesis $h_{t,p}$ in the standard lexicographical enumeration of all hypotheses with t parameters encoded using p bits precision.

Recall that a parameter value must be a number between zero and one. The complete set of possible parameter values coded in p bits is given by

$$\mathcal{P} = \{k2^{-p} | k \in \mathbb{N}, 1 \leq k \leq 2^p\}.$$

There are $\binom{2^p}{t}$ ways to pick t elements from that set. So we need only $\lceil \log \binom{2^p}{t} \rceil$ bits to code this index number.

This code is never longer than the MDL_{tp} code, as we have that:

$$\log \binom{2^p}{t} \leq \log (2^p)^t = t \log 2^p = tp. \tag{3.2}$$

The first inequality comes from the fact that:

$$\binom{x}{y} = \frac{x!}{y!(x-y)!} = \frac{x-y+1}{y} \cdot \frac{x-y+2}{y-1} \cdot \ldots \cdot \frac{x-1}{2} \cdot \frac{x}{1} \leq x^y. \tag{3.3}$$

As[3]:

$$\log \binom{m}{t} = m\mathcal{H}(t/m) + \mathcal{O}(\log m), \tag{3.4}$$

we will use in this thesis

$$\log \binom{m}{t} \approx m\mathcal{H}(t/m) \tag{3.5}$$

[2] With *optimal* we mean the number of bits yielding minimal total description length.

[3] Taken from [CT91] example 12.1.3. Think of the t-element subset as the 1's in a binary string of length m. Using the Shannon-Fano code we can describe such a string in at most $m\mathcal{H}(t/m)$ bits, see subsection 3.2.1.

. We have as a total code length for a t-parameter model:

$$L^*(p) + L^*(t) + \log \binom{2^p}{t} \approx L^*(p) + L^*(t) + 2^p \mathcal{H}(t/2^p). \tag{3.6}$$

We call MDL using this coding scheme MDL_{pss} as we essentially code the SubSet of p-precision parameters.

The difference between the bits needed by MDL_{pss} and MDL_{tp} can be quite large. Suppose we want to code 100 parameters in ten bits precision each. Using MDL_{tp} we would need $1000 + L^*(p) + L^*(t)$ bits. If we use MDL_{pss} we would need at most $473 + L^*(p) + l^*(t)$ bits. Note that $L^*(10) = 8$ and $L^*(100) = 13$.

KMDL

In [KMNR97], no optimization over parameter precision is performed. The set of possible parameter values is restricted to the set $S^x = \{x|(x,y) \in S\}$ where S is the training sample, thus restricting parameter values to the x-data. So in this coding scheme the data is not only used to *select* a hypothesis, the data is also used to *describe* a hypothesis. Using the data to describe a hypothesis leads to very short code lengths. This scheme allows consistent hypotheses. This is simply the case because we can put a switch after each example. The only thing that has to be coded, using the coding scheme that was used in [KMNR97], is which set $h_t \subseteq S^x$ is used as a hypothesis. This can be done by giving the index of h_t in an enumeration of all subsets with t elements of S^x. To do this we only need to specify t and the index number of h_t within the enumeration. We can do this in $L^*(t) + \log \binom{m}{t}$ bits, where m is the number of examples. Using Inequality 3.4, the code length is given by $L^*(t) + m\mathcal{H}(t/m) \leq m + \log t$, so m bits is roughly the maximum number of bits we need to encode a hypothesis using this coding scheme.

It is clear that the coding schemes just discussed yield quite different code lengths for some hypothesis h. It is not very clear at first sight whether or not these three different coding schemes will select hypotheses with (approximately) the same number of switches. To answer this question we have conducted several experiments to compare Two-Part MDL using each of the coding schemes. Section A.1 deals with the algorithms used to automatically generate the hypotheses that minimize the description length when we do not use the x-data to code a hypothesis.

3.3 Motivation for the Project

In this section we present a more elaborate motivation of the project.

As stated in the introduction we have four main questions that we want to answer in this project:

1. Is the coding scheme used in [KMNR97] allowed by the Two-Part MDL principle?

2. Would the experimental results of Two-Part MDL still hold when a coding scheme is used which differs from the one that was used in [KMNR97]?

3. Would the foundation of GRM still hold when it is adapted as was done in [KMNR97]?

4. How does Predictive MDL perform on the problem domain used in [KMNR97] and how is PMDL related to Cross Validation?

The first two of these goals came forth directly from the observation that MDL displayed overfitting in the experiments in [KMNR97]. This was a surprising observation because MDL is essentially trying to avoid overfitting, as we will argue after we have explained the phenomenon of overfitting.

Overfitting

We say a model selection method is *overfitting* or *overcoding* the data if the model chosen by that model selection method describes the *training* data *too precise*. In extreme overfitting the model does not capture the main structure of the data anymore, but merely repeats the data more or less accurately. Think for example of the problem of selecting the polynomial that describes a certain data set of m pairs (x, y). The model class is the set of all polynomials, so the problem is to estimate the right parameter values, as well as the right number of parameters. Suppose that all pairs are close to some polynomial with degree n, $n \ll m$. Note that for each set D such that:

$$D = \{(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)\}, \text{ where } i \neq j \Rightarrow x_i \neq x_j,$$

there is a unique polynomial f^* of degree $m-1$ such that: $\forall_{1 \leq i \leq m} : y_i = f^*(x_i)$. Selecting f^* would be called overfitting. Although this polynomial fits exactly on the data it is clear that the best fitting polynomial with degree n usually gives a much better idea of how the variables x and y are related. Furthermore, the latter model is, at least intuitively, also more likely to make good predictions of future data.

In the way we just presented overfitting we used the target function or data generating system to determine what is and what is not overfitting. In the polynomial example we used the fact that the samples were given by some polynomial of degree n. However, in the typical situation of inductive inference we do not have access to the data generating system. So, in general,we will not be able to determine whether or not we are overfitting.

MDL Tries to Avoid Overfitting

We present a simple argument why MDL is trying to avoid overfitting.

A model that overfits the data should capture a large fraction of the features of the observed data. With high probability we have that the observed data can not be described with few bits. Therefore we have with great probability that the overfitting model has to be described using many bits. We have seen in Chapter 2 that MDL makes a trade-off between model complexity and goodness of fit. We do not expect MDL to select very complex models, as we have with small probability that such a model would optimize the trade-off due to the great description length of such a model. Therefore we say that MDL is trying to avoid overfitting.

Now we give the main reasons for why we doubt whether the coding scheme that was used for MDL in [KMNR97] was correct.

What is the definition of a hypothesis?

Usually a *hypothesis*, h, is a relation over the data or a function that maps values of some domain X to some domain Y, so $h : X \to Y$. However, in [KMNR97] a 'hypothesis' is dependent on a subset of the x values of the given sample. A description of a hypothesis is given as a function $h' : X^m \to (X \to Y)$, where $X^m \subset X^*$ and X^* is the example space (elements of X^* are sequences of examples

without a labeling[4]). So, if we want to be able to determine the actual values of the parameters, we need S^x, the x values of the actual sample S, to determine which values we use as parameters. Only then can we construct a function $h : X \rightarrow Y$.

If we think of a sender-receiver situation[5] in which the hypothesis is sent, S^x should also be known to the receiver or S^x should be sent in addition to construct h from h'. However, if we have to send S^x too, the total description of S^x and h' will always be longer than a direct description of h using a data independent coding scheme. The last claim is easy to check. We look at both description lengths starting with the data independent description. Using the data-independent coding scheme we need to code, say, t parameters with some precision of, say, $\hat{p}$ bits. So we can describe the hypothesis in $t\hat{p}$ bits. Now, consider the alternative description in which we also have to code S^x. The coding of S^x would take mp^* bits, where p^* is the number of bits in which each x data item is specified. There always exists a 'traditional' hypothesis that can be described in at most mp^* bits, which is equivalent to the hypothesis that contains S^x. With a traditional hypothesis we mean a hypothesis that is described without using the data. We obtain this shorter hypothesis by simply putting switches at the same positions and describe them in p^* bits. Because there cannot be more than m switches the equivalent hypothesis will never have a longer code length than the one containing S^x. As discussed in Chapter 2, we prefer coding schemes that yield short codes over ones that yield long codes.

Learning is compression: $KMDL$ does not compress in general!

Consider the situation in which we perform model selection given some sample S_1. Now we get another sample, S_2, with different x values than S_1. Furthermore, suppose that S_1 and S_2 were generated by the same data generating system. We want to use the model selected on the basis of S_1 to code S_2 (so both the x and y-values). To do this, we need a *stand-alone* hypothesis. With *stand-alone* we mean that no additional information is needed to decode the model from the codeword. As we have seen before, the codewords of **stand-alone hypotheses** produced by the $KMDL$ coding scheme are very long as compared to the codewords assigned by the MDL_{pss} coding scheme.

Let S_1^x and S_2^x denote the x-data of respectively S_1 and S_2. As, under the uniform distribution, it is a very unlikely event that we can code S_2^x in few bits given S_1^x we expect to have:

$$(L_K(S_1^x) + L_K(H|S_1^x)) + L_K(S_2|H, S_1^x) < L_{pss}(H) + L_{pss}(S_2|H),$$

where $(L_K(S_1^x) + L_K(H|S_1^x))$ is the description length of the hypothesis for the $KMDL$ coding scheme and $L_{pss}(H) + L_{pss}(S_2|H)$ is the length of the encoding of S_2 with the MDL_{pss} coding scheme.

Furthermore, the total $KMDL$ code length for **all** other data sets, for which the x-data cannot be coded in few bits given S_1^x, will be considerably larger than the code length of the training data set[6].

Now remember the correspondence between code lengths and probability distributions described in Chapter 2. We just saw that only data sets with x-values that

[4] See page 84

[5] See Section 2.5, page 18.

[6] The x-data of the training sample, S, is incorporated in the stand-alone hypothesis and can thus be extracted from it by just giving a method to do so. This method can be coded in a number of bits independent of the size, m, of the training sample, so in $\mathcal{O}(1)$ bits.

The x-data of a different sample, S', however, is in general not incorporated into the stand-alone hypothesis and should be coded along with the stand-alone hypothesis in at least $K(S'|S)$ bits, probably resulting in a relatively long code. As the values of the x-data are coded in a precision that is at least as high as the precision used by the MDL_{pss} coding scheme.

can be coded in few bits if S_1^x is given, can get short total codes. Now suppose that we let the model selected by MDL define our guess for the probability distribution that underlies the data generating system. This distribution renders data sets, for which the x-data can be coded in few bits if S_1^x is given, more likely than other data sets. The description length, and thus also probability, of a sample is greatly dependent on the x-values. Due to this strong dependence it may be the case that some model makes the same training error $\hat{\epsilon}$ on two different data sets, although it may assign completely different code lengths and thus probabilities to the two data sets. This is a very counterintuitive result. We would like to have models that render samples equally likely, if those samples differ (in terms of error) to the same extent from the model.

Concluding:

When applying the Two-Part MDL method, we normally use the training data to **select** a hypothesis. The $KMDL$ coding scheme also uses the training data to **describe** a hypothesis. A consequence of using the training data to describe a model is that this coding of the model gives a very short description of the training data. However, if we use this coding scheme, a second sample is very unlikely to have a short code length if it is encoded on the basis of a model that was itself encoded relative to the first data sequence. So the $KMDL$ coding of a model enables us to code *one* data sequence very efficiently. A data independent coding of a model, for example the MDL_{pss} coding scheme, is likely to enable us to code *many* data sequences relatively efficiently.

The main point of critique on the use of properties of the training data to describe a hypothesis is that we obtain a coding scheme that deviates from MDL in the direction of Maximum Likelihood (ML) and thus tends to overfit the training data. Two-Part MDL, as used in the various works of Rissanen, *never* uses properties of the training data to describe hypotheses. As far as we know, no remarks are made about this possibility in the literature. In our view it is the specific problem domain that was used in [KMNR97] that has a very special property. It was probably due to the fact that this property holds for only very few problem domains, that it is not discussed in the literature about MDL. The property of the domain is:

> For every hypothesis h_t with t parameters, there is a subset $(x_{i_1}, ..., x_{i_t})$ of t elements of the x-training data that can be seen as a hypothesis h'_t with the x values in the subset as its parameters, such that:
>
> $$\hat{\epsilon}(h'_t) \leq \hat{\epsilon}(h_t)$$
>
> In other words, a subset of the x-data points is guaranteed to yield an optimal hypothesis when interpreted as parameter values.

This property can lead to overfitting when it is exploited because model complexity is becoming less dominant in the code length of the MDL Two-Part code. MDL tends toward ML in these cases, as ML is only minimizing an error term. The experiments in [KMNR97] have illustrated the latter statement. In Table 3.1 we contrast the definitions of ML and Two-Part MDL.

ML: select H_{ML} such that: $H_{ML} = \arg\min_H L(D\|H) = \arg\min_H \{-\log(P(D\|H))\} = \arg\min_H \{\hat{\epsilon}(H, D)\}$
MDL: select H_{MDL} such that: $H_{MDL} = \arg\min_H \{L(H) + L(D\|H)\} = \arg\min_H \{-\log P(H) - \log P(D\|H)\}$

Table 3.1: Maximum Likelihood as a special case of Two-Part MDL. When all models are assumed to have equal description length, Two-Part MDL reduces to Maximum Likelihood.

Chapter 4

Analysis of Experiments with Two-Part MDL

In this chapter we present, analyze and compare the experimental results that were obtained by implementing the three different coding schemes for Two-Part MDL that were introduced in the previous chapter.

4.1 Preliminaries

Applying Two-Part MDL without making use of the data to code a hypothesis, so using MDL_{tp} and MDL_{pss}, leads to a preference for simple hypotheses in the case of interval functions. First, we note the most striking difference between MDL using the sample-independent coding schemes MDL_{pss} and MDL_{tp}, and the sample-dependent coding scheme $KMDL$. Our experiments have shown that MDL_{pss} and MDL_{tp} cause MDL to select hypotheses that have approximately zero or approximately the target number of switches. The coding scheme used in [KMNR97], $KMDL$, led to maximal overcoding in regions where MDL using MDL_{pss} and MDL_{tp} preferred hypotheses with approximately zero switches. See Figure 4.1 for a plot of the performance of Two-Part MDL using the three coding schemes discussed in the previous chapter. In the remainder of this chapter we explain and analyze the behavior of Two-Part MDL using the three coding schemes.

Throughout this chapter we use d to denote the number of switches of the *target function*. With the target function we mean the function which was used to generate the data (before adding the noise!). Ultimately we would like a model selection method to give us the target function. As target functions we used interval functions with equal intervals, unless stated otherwise. So the target function $f_d(\cdot)$ divides the interval [0,1] in $d+1$ intervals. We we use t to denote the number of switches used in some hypothesis. So h_t is an hypothesis with t switches.

4.2 Preparations

In the next sections we analyze the different MDL methods and make some statements about when certain hypotheses are selected; this will allow us to explain their behavior on the target function used in [KMNR97]. To prepare our analysis, we introduce some terminology and propositions here. First, we define the *noise rate*.

Definition 4.1 *A data generating system S using some data generating function $f(\cdot)$ is said to be* **corrupted** *with* **noise rate** *η if each example x, generated by S, has a probability of η to be labeled oppositely to $f(x)$. A sample is said to be*

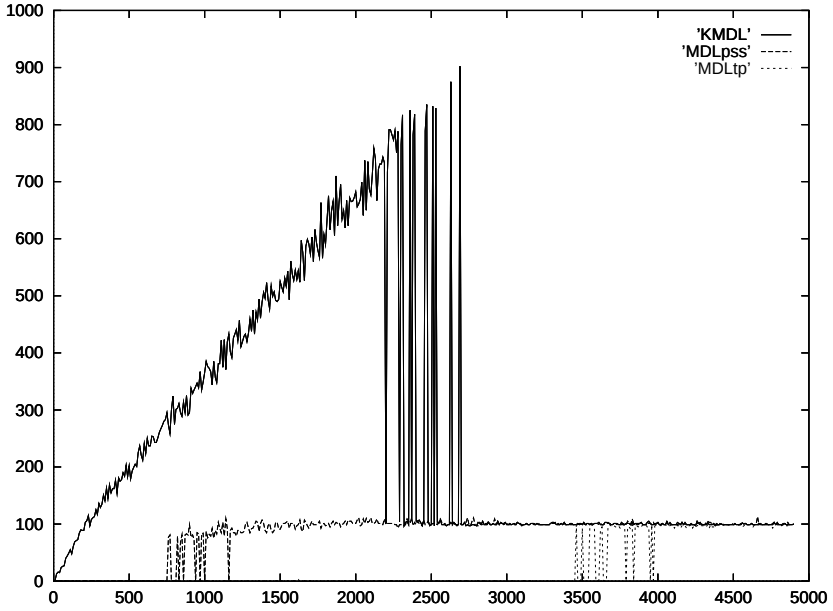

Figure 4.1: Plot of sample size, m, against hypothesis complexity, using a noise rate of $\eta = 0.2$. Note that the hypothesis complexity selected by both MDL_{tp} and MDL_{pss} is jumping from zero to $d \approx 100$ complexity almost at once.

corrupted with noise rate η if it is taken from a data generating system that is corrupted with noise rate η.

Next we define what an *optimal hypothesis* is.

Definition 4.2 *An* **optimal hypothesis** *(relative to some sample S) denoted by $\hat{h}_{t,p}$, is a hypothesis with t parameters, each of which are encoded using p bits, that has the lowest training error $\hat{\epsilon}(h_{t,p}, S)$ among all hypotheses with t parameters encoded using p bits each.*

In order to arrive at simple yet reasonably accurate theoretical analyses, we use approximations for two quantities:

1. d_0, see below, is an estimation of the expected number of switches needed in a consistent hypothesis for a sample $S(m)$ that is corrupted with noise rate η. The sample is drawn from a data generating system that uses $f_d(\cdot)$. We say that the number of switches needed in a consistent hypothesis is the number of switches in the data.

$$d_0 = 2\eta(1-\eta)m + (1-2\eta)^2 d.$$

2. $\xi(\hat{h}_t)$ is an estimation of error $\hat{\epsilon}(\hat{h}_t)$ that $\hat{h}_t$ is expected to make on a sample $S(m)$ that is corrupted by a noise rate η and was generated by a data generating system that uses an interval function with d intervals.

$$\xi(\hat{h}_t) = \begin{cases} \frac{d-t+2t\eta}{2d} & \text{if } t \leq d \\ \eta\frac{d_0-t}{d_0-d} & \text{if } t \geq d \end{cases}$$

In the rest of this section we will discuss both estimations in more detail.

Explanation of estimation of number of switches in the data

The approximation is taken from [KMNR97]. It can be explained as follows. The event that two consecutive examples belonging to the same interval are labeled oppositely happens with probability $2\eta(1-\eta)$. In the noiseless sample, there are $(m-d)$ consecutive pairs with the same label. The event that two consecutive examples, each on a different side of a switch of the target function, are labeled oppositely happens when they are both labeled correctly or both are labeled incorrectly. So this happens with probability $\eta^2 + (1-\eta)^2$; there are d such pairs. Adding the two expressions we get the approximation of the expected number of switches needed to model the sample consistently:

$$(m-d)2\eta(1-\eta) + d(\eta^2 + (1-\eta)^2) = d_0$$

Explanation of error estimation

Let d be the number of switches in the target function. We distinguish between two cases: $t < d$, undercoding, and $t > d$, overcoding.

When we make estimations of the error we make the following assumption about the noise:

> When a sample $S(m)$ of an interval function, $f_d(\cdot)$, with d switches, is corrupted by a noise rate η, we assume that $m\eta$ examples are labeled incorrectly. Furthermore, we assume that in each interval that can be defined with the switches in $f_d(\cdot)$ a fraction η of the examples in the sample is labeled incorrectly.

What we do is that we assume the mean case. There is small probability that the mean case is the actual case. However, the probability that the actual case deviates only slightly from the mean case is high. We therefore assume the mean case.[1] Now we will explain how we arrive at the quantities that are used for $\xi(\hat{h}_t)$.

Let $t \leq d$. Now approximately the best one can do using t switches, assuming the mean case, is laying the switches exactly as in the target function[2]. This will result in leaving $d-t$ intervals of the target function labeled either all positive or negative. This means that we are expected to have labeled half of examples in this region incorrectly. This region contains a fraction $\frac{d-t}{d}$ of the whole sample, so a fraction $\frac{d-t}{2d}$ of the sample is expected to be labeled incorrectly.[3]

In the intervals that are represented in the model only a fraction η of the examples is labeled incorrectly. So here a fraction $\frac{t\eta}{d}$ of the examples is labeled incorrectly. If we add the errors of both cases together, we get a total error of ξ, as given in definition 4.2 if $t \leq d$.

Suppose now that $t > d$. We know that for the $m\eta$ corrupted examples we have to use $d_0 - d$ switches to code them consistently. Now, we make an estimation of the error that we will get by using $t \leq d_0$ switches. This estimation is chosen as follows: We lack $d_0 - t$ switches to code consistently. We need $d_0 - d$ switches for the corrupted examples to be labeled consistently. There are $m\eta$ corrupted

[1] See appendix B for a more elaborate justification of this claim.

[2] This was experimentally confirmed.

[3] Here, we neglect the case in which $d-t$ is even. In this case the badly modeled area does not contain the same number of *positive intervals* and *negative intervals*. There is either one more positive interval or one negative interval. So in fact only $1/2 - \frac{1}{2(d-t+1)}$ of the examples in this region will be labeled incorrectly, as for an 'extra' fraction $\frac{1}{2(d-t+1)}$ of the area the labeling is correct. As the size of the region is given by $(d-t)/d$, neglecting the '$d-t$ is even' case, causes the total error estimation to be a fraction

$$\frac{d-t}{d} \cdot \frac{1}{2(d-t)} = \frac{1}{2d},$$

too pessimistic, which is only slightly too pessimistic.

examples. So, the fraction between the number of extra switches (over d) that are available in $\hat{h}_t$ and the number of extra switches needed for consistent modeling is $\frac{d_0-t}{d_0-d}$. As there were $m\eta$ corrupted examples, we expect to leave $m\eta\frac{d_0-t}{d_0-d}$ examples labeled incorrectly. So we will leave a fraction $\frac{\eta(d_0-t)}{d_0-d} = \xi$ of the examples labeled incorrectly.

In the next sections we explain the results of the experiments we have obtained. To do so, we will often use $\mathcal{H}(\xi(\hat{h}_t))$, for convenience we state here definitions of $\mathcal{H}(\xi(\hat{h}_t))$ and its first and second derivative, where $\mathcal{H}(\cdot)$ is the binary entropy function. First we note:

Proposition 4.1 *If $\mathcal{H}(\cdot)$ is the entropy function, then the first derivative of $\mathcal{H}(\cdot)$ is given by:*

$$\mathcal{H}'(x) = \log(1-x) - \log x.$$

Using this proposition we arrive at:

$$\mathcal{H}(\xi(\hat{h}_t)) = \begin{cases} \mathcal{H}(\frac{d-t+2t\eta}{2d}) & \text{if } t \leq d \\ \mathcal{H}(\eta\frac{d_0-t}{d_0-d}) & \text{if } t \geq d \end{cases}$$

$$\frac{\partial\mathcal{H}(\xi(\hat{h}_t))}{\partial t} = \begin{cases} \frac{1-2\eta}{2d}(\log\frac{d-t+2t\eta}{2d} - \log(1-\frac{d-t+2t\eta}{2d})) & \text{if } t \leq d \\ -\frac{\eta}{d_0-d}(\log(1-\eta\frac{d_0-t}{d_0-d}) - \log(\eta\frac{d_0-t}{d_0-d})) & \text{if } t \geq d \end{cases}$$

$$\frac{\partial^2\mathcal{H}(\xi(\hat{h}_t))}{(\partial t)^2} = \begin{cases} -\frac{(1-2\eta)^2}{4d^2\ln 2}(\frac{1}{d-t+2t\eta} + \frac{1}{d+t-2t\eta}) & \text{if } t \leq d \\ -\frac{\eta^2}{(d_0-d)^2\ln 2}(\frac{1}{1-\eta\frac{d_0-t}{d_0-d}} + \frac{1}{\eta\frac{t-d}{d_0-d}}) & \text{if } t \geq d \end{cases}$$

4.3 Analyzing Behavior of MDL_{tp}

In this section we consider the behavior of MDL_{tp} using the terminology and assumptions made in the previous section.

4.3.1 Undercoding

The code length of an optimal hypothesis $\hat{h}(t,p)$ using the MDL_{tp} coding scheme is given by :

$$L_{tp}(\hat{h}_{t,p}) = tp + m\mathcal{H}(\xi(\hat{h}_{t,p})). \tag{4.1}$$

$$\frac{\partial}{\partial t}L_{tp}(\hat{h}_{t,p}) = p + m\frac{\partial}{\partial t}\mathcal{H}(\xi(\hat{h}_{t,p})) \tag{4.2}$$

$$\frac{\partial^2}{(\partial t)^2}L_{tp}(\hat{h}_{t,p}) = m\frac{\partial^2}{(\partial t)^2}\mathcal{H}(\xi(\hat{h}_{t,p})) \tag{4.3}$$

If we look at Figure 4.1, a plot of the complexity t of the learned hypothesis versus the number of examples, m, available to the learning algorithm, we see that first the *zero-hypothesis* is selected until m exceeds some value. With the *zero-hypothesis* we mean the hypothesis that has zero switches. If m is greater than that value, the hypothesis with d switches is selected.

Suppose that the noise is distributed over the sample $S(m)$ as assumed in the previous section, so we have $m\eta$ examples that are labeled incorrectly. Furthermore, suppose that the data generating function is an interval function with d switches. Let the model class $\mathcal{M}$ consist of all models with d or fewer switches[4]: $\mathcal{M} = \bigcup_{1 \leq i \leq d} \mathcal{M}_i$. Then MDL_{tp} will select a hypothesis with 0 parameters if:

$$m < pd + m\mathcal{H}(\eta),$$

otherwise the optimal hypothesis with d parameters, each encoded using p bits, will be selected.

The previous statement can be justified as follows: Because $\frac{\partial^2}{(\partial t)^2} L_{tp}(\hat{h}_{t,p}) < 0$ for $0 \leq t \leq d$, we have that $L_{tp}(\cdot)$ has at most one extreme, which is a maximum, in this interval. So this means that either the zero-hypothesis $h_{0,0}$, or $\hat{h}_{d,p}$ yields the shortest code for an arbitrary sample. So either the zero-hypothesis or the *target hypothesis* will be selected. With the target hypothesis we mean the hypothesis that models the data generating system the best. We would generally like a learning algorithm to output the target hypothesis. In our case the data generating system is included in the model class and therefore the target hypothesis *is* the data generating system.

4.3.2 Overcoding

Overcoding will not happen as long as the consistent hypothesis yields a longer code length then the hypothesis with the right (target) number of switches d, so when Equation 4.4 holds. In the expressions below, p_0 is the minimal number of bits needed to code each parameter in the consistent hypothesis.

$$L_{tp}(\hat{h}_{d_0,p_0}) > L_{tp}(\hat{h}_{d,p}), \tag{4.4}$$

or equivalently:

$$p_0 d_0 > pd + m\mathcal{H}(\eta) \tag{4.5}$$

The latter claim can be derived from an argument similar to that in the previous subsection. Note that we have:

$$p_0 d_0 \geq p_0(d+1) \tag{4.6}$$

$$pd + m\mathcal{H}(\eta) \geq pd + m\mathcal{H}(1/m) \tag{4.7}$$

The consistent hypothesis should at least have one more switch than the target switch, otherwise there would be no overcoding at all, so $d_0 \geq d+1$. Because the parameters of $\hat{h}_{d,p}$ are coded with finite precision, it will never perform better than $\hat{h}_d$ which can have any values in $[0,1]$ as parameter values. Using the assumptions we made about the distribution of the noise among the sample, the lowest possible error of an interval function with d switches is η. So we expect $\hat{\epsilon}(h_{d,p}) \geq \eta$. Inequality 4.7 comes from the fact that the lowest possible noise rate, such that we expect to have at least one corrupted example in our sample, is $1/m$. We need to have at least one corrupted example because otherwise there is no noise present in the sample and it would be impossible to overfit the data.

We use (4.5), (4.6) and (4.7) to get:

$$p_0 d_0 - pd > (p_0 - p)d > m\mathcal{H}(1/m) \tag{4.8}$$

[4]See page 22.

The first inequality holds. So, if the second inequality holds then (4.8) holds and we do not expect overcoding to take place. In experiments (using sample sizes up to 5,000) we have learned that typical values for p_0 are around 20 bits. For a value of $m = 5,000$, we have that $m\mathcal{H}(1/m) \approx 14$. Only when we take $m = 1,000,000$, more than 20 bits are needed to code the exceptions on the hypothesis. It is clear that overcoding is never selected, because when d is small, p and p_0 are expected to differ considerably. If d is large, then this will also render the consistent hypothesis unfavorable, as d multiplies the difference $p_0 - p$ in (4.8).

Another argument for Inequality 4.8 to hold is that

$$m\mathcal{H}(1/m) < 1 + \log m,$$

this is easy to check. We also have that with high probability $p_0 > \log m$. The last claim is true because it is a rare event that a sample of length m is such that for all consecutive examples, x_i and x_{i+1}, with $y_i \neq y_{i+1}$ there is some value $x_i \leq k \cdot 2^{-\log m} < x_{i+1}, k \in \mathbb{N}^+$. Therefore, we have with high probability that:

$$p_0 + d(p_0 - p) > \log m + d(p_0 - p) > \log m + 1 > m\mathcal{H}(1/m)$$

and hence, with high probability no overcoding will take place.

4.3.3 A Lower Bound on the Number of Examples Needed to Learn Correctly

Now that we have the definitions as stated above, we can compute lower bounds on the number of examples the algorithm needs to output an hypothesis with d parameters. All we need to do is solve the inequality:

$$\begin{aligned} L_{tp}(\hat{h}_{d,p}) &< L_{tp}(h_0) \\ td + m\mathcal{H}(\xi(\hat{h}_{d,p})) &< m \\ dp + m\mathcal{H}(\eta) &< m \\ m &> \tfrac{dp}{1-\mathcal{H}(\eta)} \end{aligned}$$

So for the situation in Figure 4.1, where $d = 100$ and $\eta = 0.2$, we compute:

$$m > p\frac{100}{1 - \mathcal{H}(0.2)} \approx p \cdot 357.1$$

The optimal precision in this case appeared to be 10 bits[5] , so we get $m > 3571$ as a lower bound on the number of examples. This lower bound is supported by the conducted experiments(see Figure 4.1).

4.4 Analyzing Behavior of MDL_{pss}

This coding scheme codes the subset of parameters that is used in a certain hypothesis. If p bits are used to code parameter values, the set $\{k2^{-p} : k \in \mathbb{N}, 1 \leq k \leq 2^p\}$ is the set of all possible parameter values.

[5]The value of 10 bits was determined purely on the basis of the conducted experiments.

4.4.1 Undercoding

In this coding scheme we use only $\log \binom{2^p}{t} \leq 2^p \mathcal{H}(t/2^p)$ bits[6] to encode a hypothesis $\hat{h}_{t,p}$. Using the approximations of Section 4.2, we can define the expected code length for MDL_{pss} as:

$$\begin{aligned} L_{pss}(\hat{h}_{t,p}) &= 2^p \mathcal{H}(t/2^p) + m\mathcal{H}(\xi(\hat{h}_{t,p})) \\ \frac{\partial}{\partial t} L_{pss}(\hat{h}_{t,p}) &= 2^p \big(\log(1 - \tfrac{t}{2^p}) - \log t + p \big) + m \tfrac{\partial}{\partial t} \mathcal{H}(\xi(\hat{h}_{t,p})) \\ \frac{\partial^2}{(\partial t)^2} L_{pss}(\hat{h}_{t,p}) &= -\tfrac{2^p}{ln2}(\tfrac{1}{t} + \tfrac{1}{2^p - t}) + m \tfrac{\partial^2}{(\partial t)^2} \mathcal{H}(\xi(\hat{h}_{t,p})) \end{aligned}$$

The second derivative with respect to t is always negative. This means, as we have seen in Section 4.3.1, that $t = 0$ and $t = d$ are the only candidates to yield the shortest code length in the region $0 \leq t \leq d$. The zero hypothesis, h_0, is selected as long as:

$$\begin{aligned} L_{pss}(h_0) &< L_{pss}(\hat{h}_{d,p}) \\ m &< 2^p \mathcal{H}(d/2^p) + m\mathcal{H}(\eta) \\ m &< \tfrac{2^p \mathcal{H}(d/2^p)}{1 - \mathcal{H}(\eta)} \end{aligned}$$

So undercoding is performed until a certain value of m, if m is greater than this value the d-parameter hypothesis will yield the shortest code length.

4.4.2 Overcoding

We will use an informal argument to show why MDL_{pss} does not overcode.

The second derivative of the entropy of the expected error, $\mathcal{H}(\xi(\hat{h}_{t,p}))$, with respect to t is always negative in the region $d \leq t \leq d_0$. The same holds for the second derivative of the hypothesis length in this region. Therefore, the second derivative of the total code length is always negative in this region. So for this region only a choice between $t = d$ or $t = d_0$ has to be made. Now, we show that it is most unlikely that the consistent hypothesis will yield a shorter code length than the hypothesis with the target number of switches. There are two reasons:

1. For the consistent hypothesis we need to code (many) extra switches; $d_0 \gg d$.
2. For the consistent hypothesis we need to code the parameters using many more bits; $p_0 \gg p$.

Now we discuss both cases in more detail and we conclude with an example to illustrate.

Discussion

Note that if we get more and more examples from a source, while the noise rate remains constant, the number of switches needed for a consistent hypothesis will increase linearly in the number of examples. This is easily perceived when we consider that we expect to have $m\eta$ corrupted examples in a sample. The number of errors of the target model that has to be coded is also growing linearly in m. However, the number of bits needed to code the errors of the target model is less

[6] See Inequality 3.4, on page 24.

than one bits per error. The number of bits needed to code switches of the consistent model is considerably larger, as these switches have to be coded using high precision.

The latter claim can be justified as follows: The examples are drawn according to a uniform distribution from [0,1]. To code the switches of the consistent hypothesis we need to code numbers between each pair of examples that is labeled oppositely. We use fixed precision to code all switches, so we code all switches in the maximal precision that is needed to code all individual switches. There are only relatively few numbers in [0,1] that can be coded (using our coding scheme) that can be coded in few bits. Therefore we have that with high probability there are two neighboring examples, that are labeled oppositely, such that we need many bits to code a number between them. Thus we have with high probability that we have to use many bits to code the parameters of the consistent model.

The code length for $\hat{h}_{d_0,p_0}$ is given by $2^{p_0}\mathcal{H}(d_0/2^{p_0})$, where p_0 is the number of bits needed to code parameter values for the consistent hypothesis. It is easy to check that:

Proposition 4.2 *If $\mathcal{H}(\cdot)$ is the entropy function and $a > 1, 0 \leq x \leq 1/2$ then:*

$$a\mathcal{H}(x/a) > \mathcal{H}(x)$$

Remember that the code length of the consistent hypothesis is given by:

$$2^{p_0}\mathcal{H}(\frac{d_0}{2^{p_0}}) = 2^{(p_0-p)}2^p\mathcal{H}(\frac{d_0}{2^{p_0-p}2^p}) > 2^p\mathcal{H}(\frac{d_0}{2^p})$$

The inequality is a consequence of Proposition 4.2. The code length of the *hypothesis* (so $L(H)$ and not $L(H) + L(D|H)$) with the target number of switches is given by $2^p\mathcal{H}(\frac{d}{2^p})$. We have with high probability

$$L_{pss}(h_{d,p}) \ll L_{pss}(h_{d_0,p_0})$$

Hence, the code length for the consistent hypothesis is with high probability greater than the code length of the target hypothesis, due to the inevitable increase in precision and larger number of switches that has to be coded for the consistent hypothesis.

An Example

Now we present an example, based on experimental results, to illustrate the argument given above. We show how the high precision, needed to code the consistent model, renders the consistent model unfavorable. Take $d = 10$ and take $m = 4000$ and $\eta = 1/2000$. We expect $d_0 \leq d + 4$. In experiments still no overcoding was encountered, although d_0 is only slightly greater than d. The code lengths for the consistent hypothesis and the target hypothesis still differ about ten percent. This is mainly due to the increase in precision needed. In our experiments the consistent model would typically need about 18 bits to code switches, where the target model would use about 13 bits per switch. If we would take η much larger, so the error term of the hypothesis with d switches will increase, we have that d_0 will also be a lot bigger. This directly results in an increase of the code length $2^{p_0}\mathcal{H}(d_0/2^{p_0})$ of the consistent hypothesis, such that it yields a longer code length than the target hypothesis.

4.4.3 A Lower Bound on the Number of Examples Needed to Learn Correctly

MDL_{pss} is expected only to output hypotheses with either zero or d switches. So to find a lower bound for the number of examples that is needed to select the target

hypothesis we solve the equation:

$$\begin{aligned} L_{pss}(\hat{h}_0) &> L_{pss}(\hat{h}_{d,p}) \\ m &> \frac{2^p \mathcal{H}(d/2^p)}{1-\mathcal{H}(\eta)} \end{aligned}$$

In our experiments we found that 9 bits was the optimal precision for $d = 100$ and $\eta = 0.2$. Therefore, we compute a lower bound of $m > 1303$ for $d = 100$ and $\eta = 0.2$. Figure 4.1 roughly confirms the bound we have found.

4.5 Analyzing behavior of $KMDL$

Using the approximations introduced in Section 4.2, we define the expected code length in terms of t, m and $\xi(\hat{h}_{t,p})$ and give the first and second derivate with respect to t, the number of switches used in an hypothesis.

$$L_{KMDL}(\hat{h}_{t,p}) = m\mathcal{H}(t/m) + m\mathcal{H}(\xi(\hat{h}_{t,p}))$$

A factor m is present in the error term, as well as in the complexity penalty term. We remove it because it does not influence the minimization of the code length with respect to t. We also remove the subscript p of the hypothesis, as the parameters are coded by the x-data. So we get (with a slight abuse of notation):

$$\begin{aligned} L_{KMDL}(\hat{h}_t) &= \mathcal{H}(t/m) + \mathcal{H}(\xi(\hat{h}_t)) \\ \tfrac{\partial}{\partial t} L_{KMDL}(\hat{h}_t) &= \tfrac{1}{m}\big(\log(1-t/m) - \log t/m\big) + \tfrac{\partial}{\partial t}\mathcal{H}(\xi(\hat{h}_t)) \\ \tfrac{\partial^2}{(\partial t)^2} L_{KMDL}(\hat{h}_t) &= \tfrac{1}{t(m-t)\ln 2} + \tfrac{\partial^2}{(\partial t)^2}\mathcal{H}(\xi(\hat{h}_t)) \end{aligned}$$

4.5.1 Undercoding and Overcoding

From the definitions above we learn that for both intervals of t ($t \in \{0, .., d\}$ and $t \in \{d, .., d_0\}$), the second derivative with respect to t is negative. From this we can conclude that the code length has at most one extreme, which is a maximum in both intervals. So we can conclude that the shortest code lengths are given by values of $t \in \{0, d, d_0\}$. Note that $L_{KMDL}(h_0) = \mathcal{H}(\xi(h_0)) = \mathcal{H}(1/2) = 1$ and $L_{KMDL}(h_{d_0}) = \mathcal{H}(d_0/m)$. Combining this with the fact that $\mathcal{H}(x) \leq 1$, we see that the consistent hypothesis $\hat{h}_{d_0}$ never has a code longer than the code length of the zero-hypothesis. So, we drop the option that h_0 gives us the shortest code length. This shows that undercoding is not performed by $KMDL$.

4.5.2 A Lower Bound on the Number of Examples Needed to Learn Correctly

This subsection is divided in two parts. First we formally analyze for which noise rates $KMDL$ learns correctly. Second we look at an example that illustrates the analysis.

Formal Analysis

We follow a slightly different approach here. We don't fix d and η and then try to find values for m for which KMDL will select $\hat{h}_{d,p}$. Instead, we consider the following problem: we pick a certain d and m and we try to state for which noise

rates η $KMDL$ will come up with hypothesis $\hat{h}_{d,p}$ instead of giving us the consistent hypothesis.

If both m and d are fixed, then define $\lambda = d/m$. We can rewrite $d_0 - d$, the number of switches that we expect to need in addition to the 'right' number of switches to code a consistent hypothesis, as $2m\eta(1-\eta)(1-2\lambda)$.[7] Now we write the code length (divided by m) for $\hat{h}_t$ as:

$$L_{KMDL}(\hat{h}_t) = \begin{cases} \mathcal{H}(d/m) + \mathcal{H}(\eta) = \mathcal{H}(\lambda) + \mathcal{H}(\eta) & \text{if } t = d \\ \mathcal{H}(d_0/m) = \mathcal{H}(2\eta(1-\eta)(1-2\lambda) + \lambda) & \text{if } t = d_0 \end{cases}$$

As we let η go towards $1/2$, we see that $L_{KMDL}(\hat{h}_{d_0})$ goes to 1, where $L_{KMDL}(\hat{h}_d)$ goes to $1 + \mathcal{H}(d/m)$. So for large η the consistent hypothesis will yield shorter code lengths than the hypothesis with d switches. This is consistent with the intuition that large noise rates yield bad performances, in the sense that in such cases the algorithm will not output $\hat{h}_d$.

When we let η go to a very small value, $L_{KMDL}(\hat{h}_{d_0}) < L_{KMDL}(\hat{h}_d)$ holds, as we will show below. This means that if we have very little noise in our sample, $KMDL$ will perform worse than with some more noise. This is intuitively quite strange, because one would think that less noise would only make the algorithm perform better. We will now show that for every m and d with $d/m < 1/2$, there are values $\eta > 0$ such that $L_{KMDL}(\hat{h}_{d_0}) < L_{KMDL}(\hat{h}_d)$.

Let $\alpha = 2\eta$. We have that $d_0 \leq d + 2m\eta$, because in the worst case we would have to use two switches to code each corrupted example. Overcoding occurs if:

$$L_{KMDL}(\hat{h}_{d_0}) < L_{KMDL}(\hat{h}_d),$$

or equivalently, if:

$$\mathcal{H}(d_0/m) < \mathcal{H}(\lambda) + \mathcal{H}(\frac{1}{2}\alpha). \tag{4.9}$$

Since

$$\mathcal{H}(d_0/m) \leq \mathcal{H}(\frac{d + 2m\eta}{m}) = \mathcal{H}(\lambda + \alpha), \tag{4.10}$$

Inequality 4.9 is satisfied if:

$$\begin{aligned} \mathcal{H}(\lambda + \alpha) &< \mathcal{H}(\lambda) + \mathcal{H}(\tfrac{1}{2}\alpha) \\ \mathcal{H}(\tfrac{1}{2}\alpha) &> \mathcal{H}(\lambda + \alpha) - \mathcal{H}(\lambda) \\ \tfrac{\mathcal{H}(\frac{1}{2}\alpha)}{\alpha} &> \tfrac{\mathcal{H}(\lambda+\alpha) - \mathcal{H}(\lambda)}{\alpha} \end{aligned} \tag{4.11}$$

We have that:

$$\lim_{\alpha \to 0} \frac{\mathcal{H}(\lambda + \alpha) - \mathcal{H}(\lambda)}{\alpha} = \mathcal{H}'(\lambda) \tag{4.12}$$

$$\lim_{\alpha \to 0} \frac{\mathcal{H}(\frac{1}{2}\alpha)}{\alpha} = \lim_{\eta \to 0} \mathcal{H}'(\eta) = \infty \tag{4.13}$$

The equalities in (4.12) and (4.13) follow from l'Hôpital's Rule.

It follows from (4.11), (4.12) and (4.13) that if we fix $\lambda < 1 - 2\eta$, we have that there is an $\eta_0 > 0$ such that for $\eta < \eta_0$ overcoding will yield a shorter code length than coding with a hypothesis with d switches.

[7] See page 33.

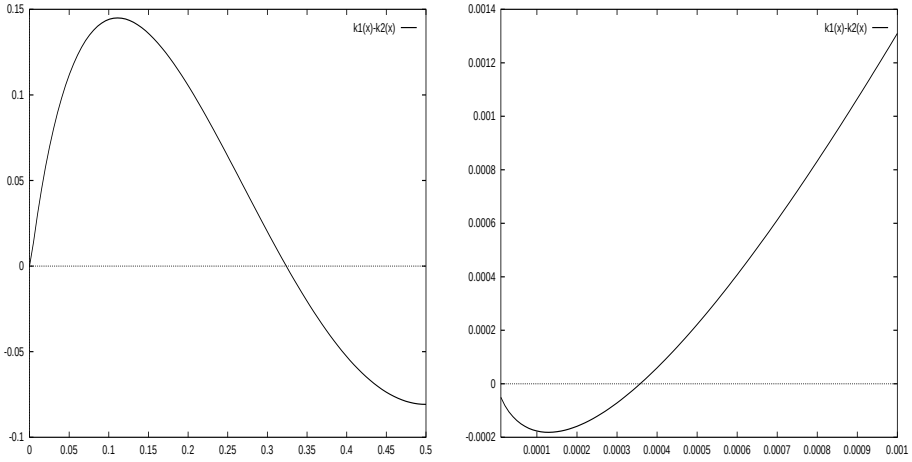

Figure 4.2: Plots of $L_{KMDL}(\hat{h}_{d_0}) - L_{KMDL}(\hat{h}_d)$ against noise rate, $\lambda = 1/100$ in this plot. When the value of this difference is larger than zero, the d parameter hypothesis will yield a shorter code length than the consistent hypothesis. For every $\lambda = d/m$ there are small noise rates that improve the behavior of $KMDL$. The right plot is a close up near zero of the left plot.

The bound on η to learn correctly given by the equation

$$L_{KMDL}(h_{d_0}) - L_{KMDL}(h_d) > 0 \tag{4.14}$$

is not very tight as the d parameter hypothesis can in general reach a lower training error than the assumed $\hat{\epsilon} = \eta$. However, what we can conclude from our experiments is that for
$\lambda = m/d < \beta$, where $\beta \approx 1/20$ there is at least one noise rate for which $KMDL$ will output the correct hypothesis. For $\lambda > \beta$, where $\beta \approx 1/20$, we have that Inequality 4.14 does not hold[8], no matter what noise rate is used.

Discussion of Experiment

Plots of the expected behavior of $KMDL$ as modeled in Inequality 4.14 are given in figures 4.2 and 4.3.

An illustration of *actual* performance that improves when more noise is added is given in Figure 4.4 on page 43. We will discuss this experiment in the remaining part of this section. For the experiment we used a target function of 100 intervals and a sample size of $m = 2500$. When a noise rate of $\eta = 0.05$ was used, KMDL (lowest curve) yields a hypothesis with 121 switches. We see that the amount of compression increases only a small amount if we use more than d switches. Both MDL_{pss} (middle curve) and MDL_{tp} (highest curve) yield a hypothesis with d switches. This is due to the increase in precision that is needed if more switches are used. The effect of the increase in precision is dramatic on the amount of compression that is achieved.

When we use a noise rate of $\eta = 0.15$, both $KMDL$ and MDL_{pss} output the $d = 100$ parameter hypothesis, while MDL_{tp} outputs the 0-hypothesis. We see that the rate of compression decreases again for the consistent hypothesis for $KMDL$ as many switches have to be coded due to the high noise rate. Due to the greater noise

[8]The value of 1/20 was determined on the basis of analysis of plots of the function $L_{KMDL}(h_{d_0}) - L_{KMDL}(h_d)$ against noise rate, see Figure 4.3.

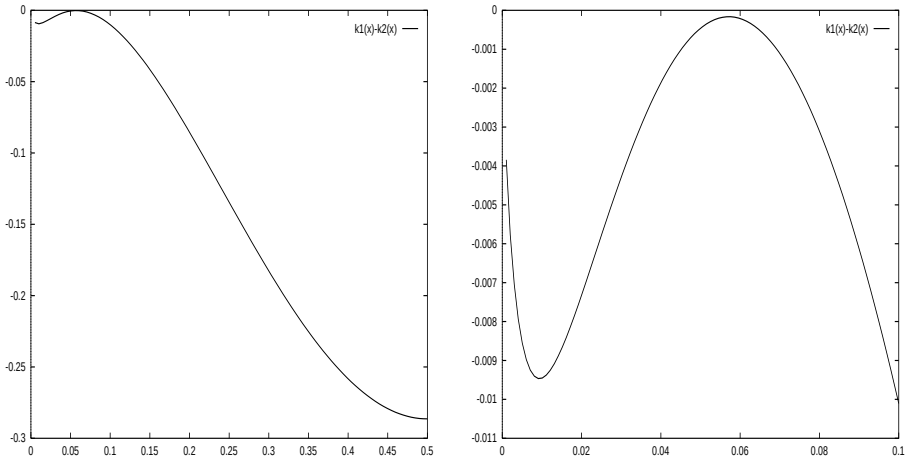

Figure 4.3: Plots of $L_{KMDL}(\hat{h}_{d_0}) - L_{KMDL}(\hat{h}_d)$ against noise rate. The right plot is a close up near zero of the left plot; it shows the region where smaller noise rates yield worse performances. In this setting $\lambda = 1/20$, we see that for all noise rates $KMDL$ is expected to overcode.

rate MDL_{tp} does not achieve compression using the target hypothesis. Therefore MDL_{tp} performs undercoding for this greater noise rate.

4.6 Comparing MDL_{pss} and $KMDL$

Now we compare $KMDL$ and MDL_{pss} in terms of generalization error. Our experimental results have shown that MDL_{pss} makes a better estimation of the complexity of the data generating function; that is: MDL_{pss} does not display the extreme overfitting that $KMDL$ displays and MDL_{pss} converges considerably faster to the correct model complexity. However, the actual *parameter estimation* of MDL_{pss} is of such poor quality that $KMDL$ outperforms MDL_{pss} in terms of *generalization error*.

The poor quality of the MDL_{pss} parameter value estimates is due to the minimization of the code length. As there are only relatively few short code words, we can only assign short code words to few parameter values. However, in general and in this problem domain in particular, the data generating machinery has no preference for the parameter values that we choose to assign short code words. The preference of Two-Part MDL for those parameter values that can be coded in few bits, is in some sense arbitrary. We could for example add some constant, c, to each coded value x, now the values $x + c$ would be preferred.

The preceding observation points in the direction that we should use MDL to estimate the model complexity, the number of parameters that is used, but maybe not to estimate the actual values for the parameters. The estimation of the parameter *values* could then be done more accurately, yielding lower error, by ML.

We conclude this chapter by mentioning Figure 4.1 on page 32 again, which displays the performance of Two-Part MDL for a specific target function. We hope that the analyses presented in this chapter explain to the reader the performance of Two-Part MDL on the interval function selection problem, using the three different coding schemes.

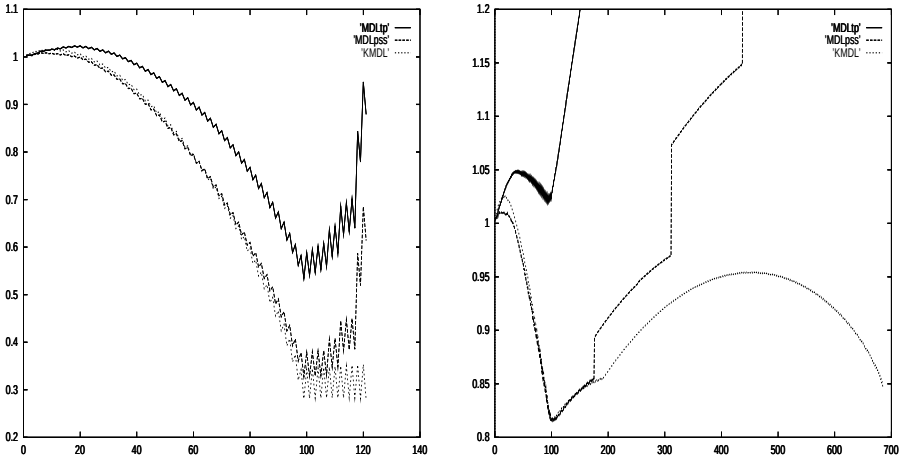

Figure 4.4: Plots of normalized (to m) code length of $KMDL$, MDL_{pss} and MDL_{tp} against the number of switches that is used in the hypothesis. In both experiments $m = 2500$ and $d = 100$. The left plot shows results for $\eta = 0.05$ and the right plot for $\eta = 0.15$. The scale on the vertical axis can be interpreted as the rate of compression of the original data.

Chapter 5

Guaranteed Risk Minimization

In this chapter we take a look at three versions of another model selection principle, Vapnik's Guaranteed Risk Minimization(GRM) and compare them to Two-Part MDL.

5.1 Preliminaries

Guaranteed Risk Minimization is also called Structural Risk Minimization [Vap82, Vap95]. It is a statistical method developed by V. Vapnik and it is extensively discussed in [Vap82]. GRM uses the *Vapnik-Chervonenkis Dimension* of a model class as a complexity measure for the models in that model class.

In [Vap82] probabilistic upper bounds[1] on $\epsilon - \hat{\epsilon}$ are formulated. The quantity $\epsilon - \hat{\epsilon}$ is the amount the generalization error differs from the training error. To put it in other words: *how much the expected performance and the actual performance differ*. As we will see below, these bounds can be used to upper bound ϵ, which we want to minimize in inductive inference.

When analyzing MDL, we looked at the behavior of the total code length $L_C(H) + L_C(D|H)$ with respect to the number of switches used by a certain hypothesis and the number of examples available to the algorithm. We call $L_C(H)$ the complexity term for Two-Part MDL, as it assigns a penalty for its complexity to a complex hypothesis. More complex models get penalized more. The complexity term can be seen as to compensate for a low training error, $\hat{\epsilon}$; a very complex model may fit the data very well ($\hat{\epsilon}$ is low), but such a model is likely to overfit the data (in which case we may have $\hat{\epsilon} < \epsilon$). We therefore add a penalty term to $\hat{\epsilon}$ for complex models. This way the complex model is rendered less favorable and we hope to avoid overfitting.

In this chapter on GRM we will use slightly different model classes than we did before, because the theory developed in [Vap82, Vap95] assumes so. We will use model classes $\mathcal{M}_{\overline{k}}$ that are defined as follows:

$$\mathcal{M}_{\overline{k}} = \bigcup_{1 \leq i \leq k} \mathcal{M}_i,$$

where $\mathcal{M}_i$ is defined as before: the set of all interval functions with i intervals:

$$\mathcal{M}_{\overline{1}} \subset \mathcal{M}_{\overline{2}} \subset ... \mathcal{M}_{\overline{d_0 - 1}} \subset \mathcal{M}_{\overline{d_0}} \tag{5.1}$$

[1] By probabilistic bounds we mean bounds that hold with a certain probability.

5.2 Standard GRM

Now, we present GRM as it was introduced in [Vap82] for infinite model classes. First, we need to define the Vapnik-Chervonenkis dimension (VC dimension). We do so by using the concept of Growth Function. So let us first introduce this Growth Function[2]. We write $\Pi_{\mathcal{M}}(S)$ as the number of distinct classifications of some sample S, that can be obtained using models in $\mathcal{M}$. Note that a sample in this context does not contain a labeling of the examples as in our definition of a sample on page 22.

Definition 5.1 *The* **Growth Function** $\Pi_{\mathcal{M}}(\cdot)$ *is defined as the maximal possible number of different classifications, by models from some model class* $\mathcal{M}$*, of a sample of a given length* m*. Let* X^* *be the sample space and let* $X^m \subset X^*$ *be the set of all samples of length* m*.*

$$\Pi_{\mathcal{M}}(m) = \max\{\Pi_{\mathcal{M}}(S) : S \in X^m\},$$

If $\Pi_{\mathcal{M}}(m) = 2^m$, it means that there is a sample $S(m)$ that can be labeled in any possible way by models in $\mathcal{M}$. We say $\mathcal{M}$ **shatters** $S(m)$.

Definition 5.2 *The* **VC dimension**[3] *of a model class* $\mathcal{M}$*, denoted by* $VCD(\mathcal{M})$*, is defined as the maximum* m *for which we can draw some sample* $S(m) \in X^m$ *that is shattered by* $\mathcal{M}$*:*

$$VCD(\mathcal{M}) = \max\{m : \Pi_{\mathcal{M}}(m) = 2^m\}$$

Sometimes we write $VCD(h)$, where h is some model. With this notation we mean $VCD(\mathcal{M}_{\overline{i}})$, such that:

$$i = \min\{k : h \in \mathcal{M}_{\overline{k}}\}$$

Note that the size of the model class $\mathcal{M}$ is bounded from below by $2^{VCD(\mathcal{M})}$.

GRM uses the following evaluation function[4]:

$$GRM(h_t) = \hat{\epsilon}(h_t) + \frac{2VCD(h_t)(\ln(\frac{2m}{VCD(h_t)}) + 1)}{m}\left(1 + \sqrt{1 + \frac{\hat{\epsilon}(h_t)m}{VCD(h_t)(\ln(\frac{2m}{VCD(h_t)}) + 1)}}\right), \tag{5.2}$$

where m is the number of examples available to the learner and h_t is the evaluated hypothesis. In the interval function selection problem the VC dimension of $\mathcal{M}_{\overline{d}}$ is d. This is easy to see. If we have a sample $S(m), m > d$, we cannot label it such that $l_i \neq l_{i+1}, 1 \leq i \leq m-1$. So $VCD(\mathcal{M}_{\overline{d}}) \leq d$. Because there is a hypothesis in $\mathcal{M}_{\overline{d}}$ which can divide $[0,1]$ into d or fewer arbitrary intervals, $\mathcal{M}_d$ shatters samples $S(m)$ with $m = d$ and for $1 \leq i, j \leq m : i \neq j \Rightarrow x_i \neq x_j$. So we can replace the term $VCD(h_t)$ in (5.2) by t; as in (5.3)

$$GRM(h_t) = \hat{\epsilon}(h_t) + \frac{2t(\ln(2m/t) + 1)}{m}\left(1 + \sqrt{1 + \frac{\hat{\epsilon}(h_t)m}{t(\ln(2m/t) + 1)}}\right). \tag{5.3}$$

[2] The definitions of the Growth Function differ. Sometimes, as in [AB92], it is defined as the maximal number of classifications that can be made using models in some model class on some sample from the sample space. Others define it as the natural logarithm of this quantity, as in [Vap95]. We will use the definition as in [AB92].

[3] See for example [AB92] Chapter 7 for a more extensive discussion of the VC dimension.

[4] See [Vap82] or [Vap95] for how this function is obtained.

5.3 An Adapted Version of GRM

In [KMNR97] an adapted version of GRM is analyzed and compared to Two-Part MDL and Cross Validation. It is remarked in [KMNR97] that GRM as it was stated by Vapnik in [Vap82, Vap95] is not quite competitive with the two other model selection methods that are evaluated in the paper. This appears to be the only motivation for the choice to adapt GRM as mentioned in [KMNR97].

In this section we are concerned with this adaptation. First we will discuss the adaptation. In Subsection 5.3.2 we argue that the adaptation implies using a *finite* model class. However, the adapted rule originated from the GRM model selection rule for *infinite* model classes. We evaluate performance of GRM using both finite and infinite model classes. To do so we define two finite model classes in Subsection 5.3.3.

5.3.1 The Modified Rule

The adaptation is to replace the term

$$VCD(h_t)(\ln \frac{2m}{VCD(h_t)} + 1) = t(\ln (2m/t) + 1)$$

in Equation 5.2 by $VCD(h_t)$ and to divide the complexity term by 2. This causes the complexity term to decrease, so the new version of GRM will in general select more complex models than the original version. We will refer to the adapted version as $KGRM$. The evaluation function for $KGRM$ is given by (recall that $VCD(h_t) = t$):

$$KGRM(h_t) = \hat{\epsilon}(h_t) + (t/m)(1 + \sqrt{1 + \hat{\epsilon}(h_t)m/t}) \tag{5.4}$$

5.3.2 Problems with the Adaptation

In this subsection we argue why Equation 5.4 as a model selection rule cannot be justified in terms of the theory of GRM presented in [Vap82, Vap95]. Thus, we argue for a negative answer to the question whether $KGRM$ can be justified as a version of Vapnik's GRM. The following lemma enables us to show why Equation 5.4 is not allowed by GRM.

Lemma 5.1 *Let $\mathcal{M}_{\overline{t}}$ be defined as on page 45. $\Pi_{\mathcal{M}_{\overline{t}}}(m) > m$, and hence cannot be upper bounded independently of m.*

Proof The smallest model subclass $\mathcal{M}_{\overline{1}} \subseteq \mathcal{M}_{\overline{t}}$ contains infinitely many interval functions that divide the domain into two intervals. So the growth function $\Pi_{\mathcal{M}_{\overline{1}}}(\cdot)$ would depend on m because for each sample size m we could pick a sample such that we can label it in m ways. We can repeat the same argument for models with more than two intervals. We can not upper bound $\Pi_{\mathcal{M}_{\overline{t}}}(m)$ independently of m, as for every upper-bound β we can find an m such that $\Pi_{\mathcal{M}_{\overline{t}}}(m) > \beta$. □

The term $VCD(\mathcal{M})(\ln \frac{2m}{VCD(\mathcal{M})} + 1)$ appears in (5.2) because it is a general upper bound on the natural logarithm of the growth function, $\ln \Pi_{\mathcal{M}_{\overline{t}}}(2m)$.[5] Using (5.4) instead of (5.2) can only be justified if t would also be a general upper bound on $\ln \Pi_{\mathcal{M}_{\overline{t}}}(2m)$. This is impossible, as shown by Lemma 5.1. Hence $KGRM$ cannot be justified as a version of GRM.

However, if, instead of the *infinite* model class $\mathcal{M}_{\overline{t}}$, a *finite* model class is used, then t may be used as an upper bound on the natural logarithm of the growth

[5] See [Vap82] or [Vap95].

function after all. In the next subsection we show that, instead of considering the full class $\mathcal{M}_{\overline{t}}$, one may also decide to restrict the search for a good model to a finite subclass of $\mathcal{M}_{\overline{t}}$. It seems that then using t as an upper bound on the natural logarithm of the growth function (and hence using (5.4)) can be justified after all. Alas in that case, we are dealing with finite model classes and, according to Vapnik[Vap82, Vap95], an entirely different formula (different from both (5.2) and (5.4)) applies. This formula will be discussed in Section 5.4.

5.3.3 Three Model Classes

Now we show for which model class t does give a reasonably good upper bound for the natural logarithm of the growth function. To do so we introduce model class $\mathcal{M}'$.

Suppose that model class $\mathcal{M}'_{\overline{t}}$ contains just one model with t parameters. Furthermore, suppose that all other models in $\mathcal{M}'_{\overline{t}}$, with fewer than t parameters have parameter values that are also parameter values for the model with t parameters. If the order of the parameters does not matter we can identify all models in $\mathcal{M}'_{\overline{t}}$ with subsets of parameter values of the t-parameter model. If this were the case there would be at most 2^t models in the model class $\mathcal{M}'_{\overline{t}}$, as a set of t elements has just 2^t subsets. Now we can conclude that t would be an upper bound on the natural logarithm of the growth function, as $\ln 2^t = t \ln 2 \leq t$. This is bound we wanted for our model class, as it the bound used in the adapted version.

The interval function selection problem has the property just mentioned, the parameters do not have to be ordered. If we regard the x-data as given then we also have that the set of parameter values of some optimal model contains the parameter values of all optimal model with fewer parameters[6].

If we have the y-data available we can shrink $\mathcal{M}'$ even further. We can compute for every number of parameters t the optimal (ML) model. This leads to a size t of model class $\mathcal{M}''_{\overline{t}}$. The growth function for $\mathcal{M}''_{\overline{t}}$ is upper bounded by $\ln t$.

The last model class seems a strange choice, as it is totally dependent of the sample. However, when we use the other two model classes, in practice we still evaluate only the models in the third class, as the models in this class are the optimal models of the other classes.

Concluding we state:

1. If we use both the x-data and the y-data to select our model classes we can upper-bound the growth function of $\mathcal{M}''_{\overline{t}}$ by $\ln t$. The size of $\mathcal{M}''_{\overline{t}}$ is finite and equal to t.

2. If we only use the x-data to select our model classes we can upper-bound the growth function of $\mathcal{M}'_{\overline{t}}$ by the general upper bound on the natural logarithm of the growth function we mentioned in Section 5.3.2, so by: $t(\ln \frac{m}{t} + 1)$. The size of $\mathcal{M}'_{\overline{t}}$ is finite and equal to $\sum_{i=0}^{t} \binom{m}{i}$. The natural logarithm of the model class size is upper bounded by the upper bound on the growth function.

3. If we do not restrict the model classes, we just take $\mathcal{M}_{\overline{t}}$ to be the set of all interval functions with t or less switches. The growth function is upper-bounded by the general bound on the natural logarithm of the growth function, so by $t(\ln \frac{m}{t} + 1)$. The size of the model class $\mathcal{M}_{\overline{t}}$ is infinite.

The next section discusses the GRM model selection rule for finite model classes.

[6] See Lemma A.1 on page 85.

5.4 GRM for Finite Model Classes

In this section we discuss yet another version of the GRM model selection rule. Foundations for this rule can be found in Chapter 4 of [Vap95]. In this publication various forms of the GRM model evaluation rule are given for various types of models and model classes. The GRM version of Equation 5.2 is meant for infinite model classes. However, as we have seen, when we make use of the x-data or both x and y-data to restrict our model classes we do not have infinitely many models in each model class $\mathcal{M}_{\overline{t}}$. The size of each model class became dependent on the sample size.

Now we consider GRM for finite model classes as it was introduced by Vapnik and derive the model selection rules for the second and third model class introduced in the previous section. The same idea as with GRM for infinite model classes applies: we have a probabilistic upper bound on ϵ, the generalization error; by minimizing this bound we select a model. However, for finite model classes GRM uses the natural logarithm of the size of the model class, instead of its VC dimension, in the computation of the complexity terms of models in the class. By the theory presented in [Vap95] an upper bound, for finite model classes, on the generalization error, $\epsilon(h_t)$, is given by:

$$\epsilon(h_t) \leq \hat{\epsilon}(h_t) + \frac{\ln N - \ln \delta}{2m}\left(1 + \sqrt{1 + \frac{4m\hat{\epsilon}(h_t)}{\ln N - \ln \delta}}\right), \tag{5.5}$$

where N is the size of the model class and δ is the confidence parameter. This means the following: for all fixed $0 < \hat{\epsilon}(h_t) < 1$ and $0 < \delta < 1$, (5.5) holds with probability $1 - \delta$. Next we derive the new evaluation functions for the model classes $\mathcal{M}'$ and $\mathcal{M}''$.

For $\mathcal{M}'$ the confidence term in Inequality 5.5 is negligible as compared to $\ln N$. Therefore, we remove it from the evaluation function. Using the model class size we computed in the previous section we arrive at:

$$\epsilon(h_t) \leq \hat{\epsilon}(h_t) + \left(\frac{t(\ln \frac{m}{t} + 1)}{2m}\right)\left(1 + \sqrt{1 + \frac{4m\hat{\epsilon}(h_t)}{t(\ln \frac{m}{t} + 1)}}\right) \tag{5.6}$$

If we use both x-data and y-data, so model class $\mathcal{M}''$, we have that $-\ln \delta$ is of about the same order as $\ln N$. Therefore we cannot just remove it from the evaluation function. The evaluation function becomes:

$$\epsilon(h_t) \leq \hat{\epsilon}(h_t) + \frac{\ln t - \ln \delta}{2m}\left(1 + \sqrt{1 + \frac{4m\hat{\epsilon}(h_t)}{\ln t - \ln \delta}}\right) \tag{5.7}$$

We have decided to test several choices for the confidence parameter. First we tested three fixed values for δ; 1/10, 1/100 and 1/1000. Second we also conducted experiments in which we let δ depend on the number of switches t in the hypothesis h_t. We used $\delta = (10t)^{-1}$. Third we also implemented a version in which we completely ignored the confidence term. In all cases the model selection behavior was the same; the consistent model was selected for all sample sizes and noise rates we tested.

In the next section we compare the four versions of GRM that we have discussed: GRM using all interval functions, $KGRM$ and the two GRM versions for finite model classes given by Equation 5.6 and 5.7.

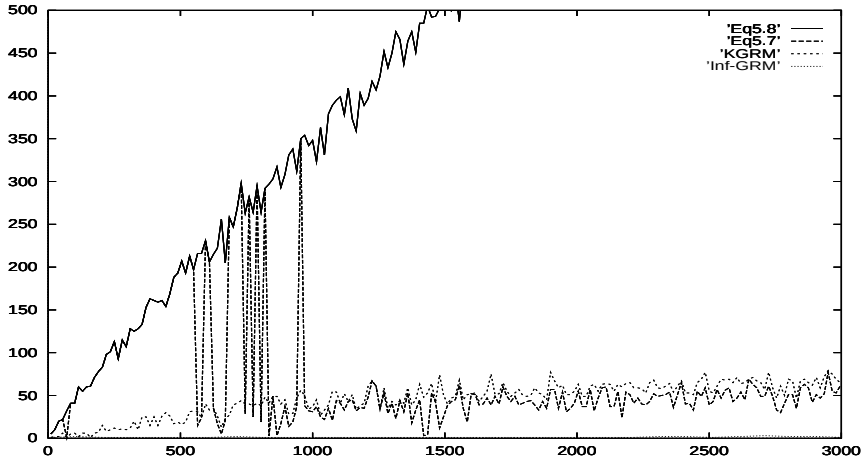

Figure 5.1: Performance of the four GRM variants discussed in this chapter. A target function with 100 unequal intervals and a noise rate of 20% were used. The upper curve shows results when Equation 5.7 was used. The curve that collapses between $m = 500$ and $m = 1000$ represents performance when we use Equation 5.6. The slowly and gradually increasing line gives the performance of $KGRM$. The dotted line just above the horizontal axis (hardly visible) gives performance of GRM for infinite model classes.

5.5 Comparison

In this last section we discuss the behavior of the GRM variants we have seen. Furthermore, we compare the behavior of $KGRM$ with that of MDL_{pss}.

5.5.1 GRM Variations

Now, we take a look at the different versions of GRM that we have been discussing. In Figure 5.1, the performance of the variants is shown on a target of 100 intervals of unequal size. Unequal intervals were chosen to make a comparison with Figure 5.4 possible. The size of the intervals was chosen at random, that is: the switches of the target function were determined by drawing 100 numbers from [0,1] by means of a uniform distribution. We see that, similar to $KMDL$, the versions of GRM for finite model classes tend to overcode for certain sample sizes. The GRM version of Equation 5.7 always picks the consistent hypothesis. The model class corresponding to Equation 5.7 is restricted by using both x and y-data; the result is maximal overfitting.

The performance of GRM using only the x-data to shrink the model class is very similar to the performance of $KMDL$. First, there is a period of overcoding in which the consistent hypothesis is selected. After this period, GRM approaches the target number of switches smoothly. We see that $KGRM$ smoothly approaches the target number switches, even for sample sizes for which GRM using (5.6) performs overcoding. GRM using the infinite model class of all interval functions yields selection of the model with zero to three switches, even for samples of length 3000, this is due to the relatively high complexity term of (5.2).

5.5.2 MDL vs GRM

First, we continue our comparison of $KMDL$ and GRM using Inequality 5.6. We plotted the behavior of $KMDL$ and GRM using Inequality 5.6 in Figure 5.2. Both

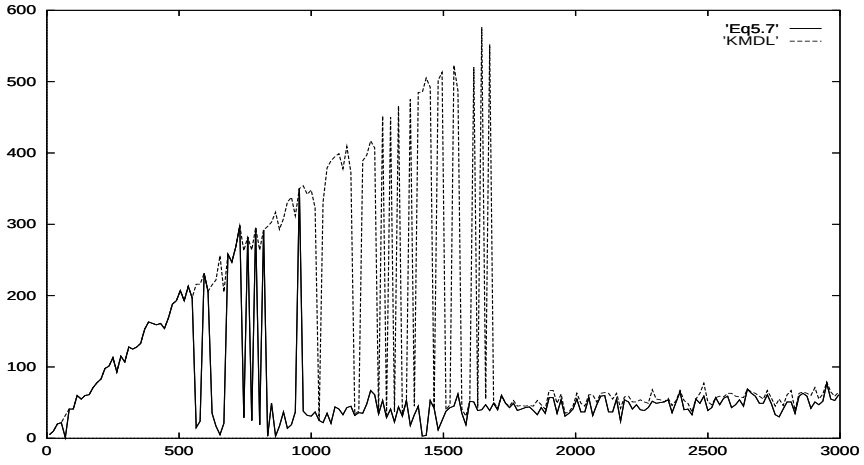

Figure 5.2: Performance of the $KMDL$ method and GRM using Equation 5.6. A target function with 100 unequal intervals and a noise rate of 20% were used. The solid line represents the behavior of $KMDL$.

methods use the x-data to shrink the model class. We see that both methods have a region from zero to some threshold value of examples in which they select the consistent model. After the threshold value, they quickly approach the correct number of switches. The GRM variant obviously performs better, as it stops over-coding for smaller m than $KMDL$.

Next we note the resemblance between the behavior of $KGRM$ and MDL_{pss}. As we have seen in the previous chapter, both MDL_{tp} and MDL_{pss} will only select models with either approximately zero or approximately the target number of switches, we call this *jumping* behavior. $KGRM$ does not perform jumping behavior.

The jumping behavior of MDL_{pss} and MDL_{tp} can be explained as follows. Each added switch in a model increases performance in terms of the training error, an equal quantity, if the total number of switches is not bigger than the target number of switches. Furthermore, the number of bits needed to code the errors decreases faster if the error is smaller, because it is given by the entropy function of the error. The number of bits needed to add a switch decreases as more switches are added. So, if adding i switches is profitable, then adding $i + 1 < d$ switches is also profitable, where d is the target number of switches. We conclude that the jumping behavior of MDL on this problem domain is not typical for MDL, but a consequence of using equal intervals in the target function. See Figure 5.3 for an illustration.

We have also conducted some experiments in which we used randomly chosen interval functions to generate the data. Some of the results are plotted in Figure 5.4. The model selection behavior of MDL_{pss} and $KGRM$ was practically identical using the random intervals. Analysis of both evaluation functions shows that both terms are indeed very much alike when we compare them as functions from number of switches to $\mathbb{R}$ for a fixed number of examples. It is, however, difficult to show this formally.

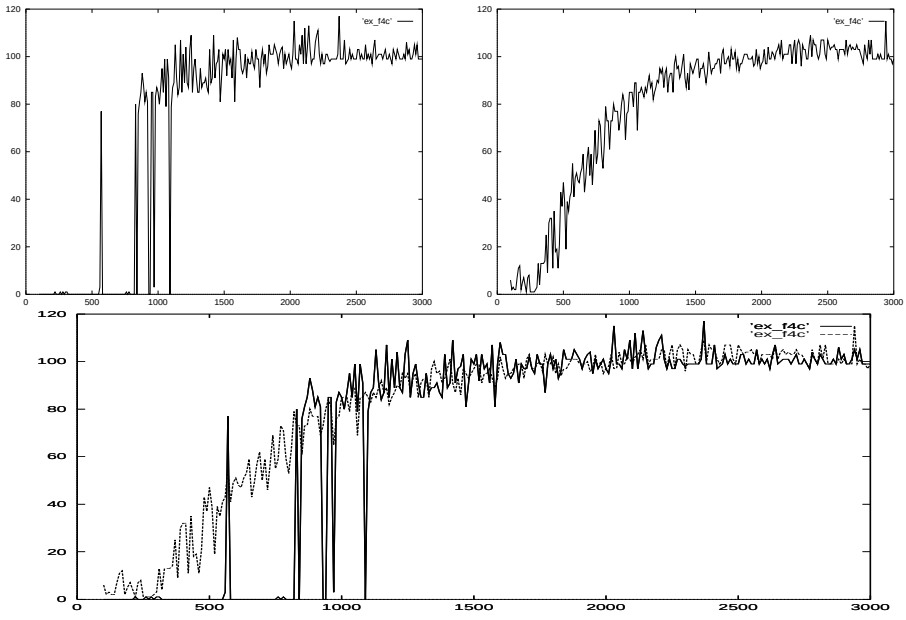

Figure 5.3: Plots of hypothesis complexity against sample size for MDL_{pss} (up left) and $KGRM$ (up right) on the same sample. A data source with $d = 100$ alternating intervals was use; the sample was corrupted by a noise rate of $\eta = 0.2$. Note that the rate of convergence to the correct number of switches is practically the same for both methods. This is shown in the bottom plot, where the dotted line represents $KGRM$.

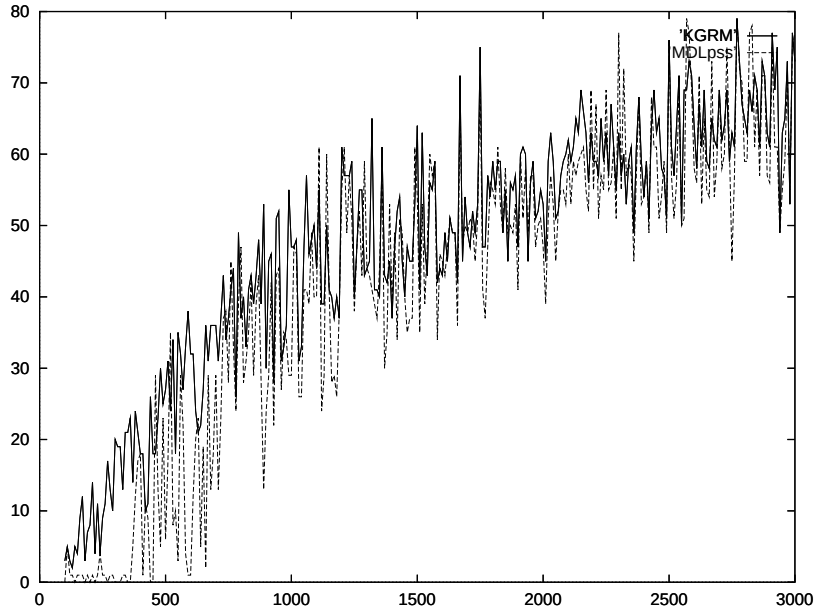

Figure 5.4: Performance of MDL_{pss} and $KGRM$. A target function with 100 unequal intervals and a noise rate of 20% were used.

Chapter 6

Predictive MDL

In this chapter we look at another way of applying the MDL principle. Rissanen's Predictive MDL is extensively treated in [Ris89]. The Predictive MDL (PMDL) technique uses the idea that models obtained from the past can be used to predict or describe the future. To make this more concrete: first, we *predict* the first observation on the basis of a-priori knowledge and next we *code* the first observation on the basis of our prediction. Then, we take the Maximum Likelihood (ML) model of the previous (first) observation(s) to *predict* the next (second) observation. We then *code* the next (second) observation using this prediction, and so on.

First, we will revisit Two-Part MDL briefly to introduce the concept of Stochastic Complexity. Subsequently, we introduce the Predictive MDL coding scheme. Finally, we take a look at the performance of PMDL on the interval function selection problem.

6.1 Two-Part MDL and Stochastic Complexity

In this section we revisit Two-Part to motivate the introduction of the concept 'Stochastic Complexity'.

Two-Part MDL Revisited

In Chapter 2, we extensively discussed Two-Part MDL. Remember that the Two-Part MDL coding scheme in fact consisted of two coding schemes, one to code the model and one to code observations with the help of a model. So, before applying Two-Part MDL we should fix two coding schemes, in order to compute the code lengths induced by those coding schemes. It became clear that one coding scheme is preferable over the other if it is less redundant. However we also saw that an optimal coding scheme that, to each object x, assigns a codeword of length $K(x)+c$, where $K(x)$ is the prefix Kolmogorov complexity of the object x and c a constant independent of x, is a non-computable coding scheme. We therefore were satisfied with coding schemes that provide us with codes that are somewhat redundant.

Now, consider the Two-Part coding scheme again. The codes are constructed in such a fashion that each observable data sequence can be coded with each model. There are in general many codewords for each possible set of data. From this fact we can conclude that the Two-Part coding scheme is inherently redundant, as we only need to code each data set in one single way. We would like to have a coding scheme that assigns just one codeword to each possible data set, so that the other codewords can be used for other datasets. The redundancy of the Two-Part coding scheme becomes also apparent when we think about what we are actually coding when using a Two-Part coding scheme. We wanted to code some data x with the

use of some model class $\mathcal{M}$ and we ended up coding both the data and a model $M \in \mathcal{M}$. As we will see, there is in general no need to explicitly code a model when we want to code some data x with the help of some model *class* $\mathcal{M}$.

Stochastic Complexity

Now, we present a way to remove some of the redundancy from the Two-Part coding scheme. We do so by defining a new probability distribution $P_{SC}(\cdot)$ on the set of possible data X.

$$P_{SC}(x) = \sum_{M \in \mathcal{M}} 2^{-L_M(x)} \tag{6.1}$$

Where $x \in X$ and $L_M(x)$ defines the length of the Two-Part codeword for x using model $M \in \mathcal{M}$. So $L_M(x) = L_{C1}(M) + L_{C2}(x|M)$, the sum of the code length for M and the code length for x using M. If we interpret the Two-Part code once again as one code, it is easy to see that $P_{SC}(\cdot)$ indeed defines a probability distribution. Just remember that prefix-free codes satisfy the Kraft Inequality[1]. More formally:

$$\sum_{x \in X} P_{SC}(x) = \sum_{x \in X} \sum_{M \in \mathcal{M}} 2^{-L_M(x)} = \sum_{x \in X} \sum_{M \in \mathcal{M}} 2^{-L(M)} 2^{-L(x|M)}$$

$$= \sum_{x \in X} \sum_{M \in \mathcal{M}} P_1(M) P_2(x|M) = \sum_{x \in X} P_2(x) = 1$$

Now, we can employ a new coding scheme that assigns a codeword of length $-\log P_{SC}(x)$ to each $x \in X$. The **Stochastic Complexity**[2] $I(x|\mathcal{M})$ of an object x relative to some model class $\mathcal{M}$ is defined as this code length:

$$I(x|\mathcal{M}) = -\log P_{SC}(x) = -\log \sum_{M \in \mathcal{M}} 2^{-L_M(x)} = -\log \sum_{M \in \mathcal{M}} P_M(x), \tag{6.2}$$

where $P_M(\cdot)$ is the probability distribution over X associated with model M. It is easy to check that for all x the length of the codeword assigned to x, given by Equation 6.2, is shorter than the shortest Two-Part code for x. Stochastic Complexity enables us to compare the modeling quality of different model *classes*, where we used Two-Part MDL to compare only *individual* models.

In the next section we will see how PMDL supplies another way to avoid the redundancy in the Two-Part MDL code.

6.2 An Introduction to Predictive MDL

The Predictive MDL technique was introduced in [Ris89]. The same technique was also discovered by Dawid[Daw84], who called it *prequential forecasting*. However, Dawid did not link the idea to code lengths.

---

[1] See page 13.

[2] As it is defined in [Ris89], page 59. In [Ris96] a different definition of Stochastic Complexity was given. It is defined by

$$I(x|\mathcal{M}) = -\log \frac{\hat{P}(x)}{\sum_{x \in X} \hat{P}(x)},$$

where $\hat{P}(x)$ is the probability distribution associated with the Maximum Likelihood model of x in $\mathcal{M}$.

Let

$$\mathcal{M} = \bigcup_{0 \leq i \leq k_{max}} \mathcal{M}_i$$

be a model class, where $\mathcal{M}_i$ is the model class of all models that have i parameters in some family of models; k_{max} may take any positive integer value. For example, if we let $\mathcal{M}$ be the set of all polynomials with 10,000 (or some other number) or fewer coefficients, then $\mathcal{M}_2$ is the set of all straight lines.

Suppose that we are given some ordered sample S of length m. Let $\hat{M}_k(x^t)$ be the Maximum Likelihood model with k parameters for the string x^t of the first t examples of $S(m)$. Remember that we can identify codes, models and probability distributions. Let $P_{\hat{M}_k(x^t)}(\cdot)$ be the probability distribution associated with $\hat{M}_k(x^t)$: then $P_{\hat{M}_k(x^t)}(\cdot)$ assigns the highest probability to x^t of all models in $\mathcal{M}_k$. Thus $P_{\hat{M}_k(x^t)}(\cdot)$ minimizes $-\log P_M(x^t)$ over all $M \in \mathcal{M}_k$. We do not want to encode the model itself that we use to code the data. Because, as we have seen, this leads to redundant codes. However, we cannot use the ML model of x^t to code x^t in the sender-receiver model without sending $\hat{M}_k(x^t)$, because we would need x^t to find $\hat{M}_k(x^t)$. More specifically, we cannot even code the single value x_t using $\hat{M}_k(x^t)$ without sending the model, as x_t is also needed for finding $\hat{M}_k(x^t)$.

The idea of PMDL is to use $\hat{M}_k(x^t)$ to code the next example x_{t+1}. We hope that the sequence of examples x^t contains some information on the example x_{t+1}, which in the end is the assumption that is behind all inductive inference. We use the past sequence x^t to code x_{t+1}, resulting in a code length for x_{t+1} of $L(x_{t+1}) = -\log P_{\hat{M}_k(x^t)}(x_{t+1})$. This leads to a total code length for a sample $S(m)$ of length m of:

$$L_{\mathcal{M}_k}(S(m)) = -\sum_{t=0}^{m-1} \log P_{\hat{M}_k(x^t)}(x_{t+1}) \tag{6.3}$$

Now we can compute $L_{\mathcal{M}_k}(S(m))$ for all $\mathcal{M}_k$ that we are interested in, and we call model class $\mathcal{M}_i$ the best, where $i = \arg\min_j\{L_{\mathcal{M}_j}(S(m))\}$. It turns out ([Ris89]) that the PMDL code length can be used as an approximation for the Stochastic Complexity.

One of the advantages of PMDL is that we do not need to specify parameter values in the coding of the data. This results in selecting only the **degree** k of the model class by applying the PMDL principle and within the optimal model class $\mathcal{M}_i$ we simply pick the ML model. This prevents poor parameter estimation we encountered when using Two-Part MDL (this will be explained on page 75). Two-Part MDL was used for model selection both within and between model classes, in other words for estimating the number of parameters as well as the value of the parameters.

PMDL, however, has some drawbacks. The most important one is the possibility that the probability for the next example, $P_{\hat{M}_k(x^{t-1})}(x_t)$, is zero or one. In fact, this will always be the case for the second symbol when using some model classes, as the Maximum Likelihood model of one example could be (if available in the model class) a probability distribution concentrating all probability on that specific outcome. The result of such probabilities is that we would compute code lengths of zero or infinitely many bits. No adequate and satisfying solution to this problem is known yet. How we work around this problem in this thesis is pointed out in the next section.

A second issue we want to point out here is that the code length of PMDL is dependent on the ordering of the data. We can order the m examples in $m!$ ways, so we would have to evaluate all the $m!$ code lengths to find the shortest code that

can be reached with the PMDL coding scheme. Certainly when small samples are considered, the ordering of the data may substantially affect the PMDL code length. In order to obtain experimental results that were more order independent, we have also implemented a version of PMDL that averages its results obtained for different orderings of the data. We also experimented with PMDL using just one ordering to get an idea of the order dependency.

In the next section we show how we applied the PMDL technique to the interval function selection problem.

6.3 Applying PMDL to the Problem Domain

As we have already seen in the discussion of Two-Part MDL, the model class that we use, as defined in Equation 3.1, only contains distributions that give zero or unity probability on the domain $Y = \{0, 1\}$ for each given x. In the previous section we have seen that this is not what we want when using Predictive MDL, because we would get codewords of length zero and infinitely many bits. However, by taking another expression than Equation 6.3 to evaluate the model classes we can still use the PMDL idea. Because our model class consists of indicator functions we can not use $-\log p(x)$ as an expression for the optimal (minimal expected) code length. Instead, we *can* code the y-data of the sample $S(m)$ efficiently by giving a bit string where a '1' on place i indicates that the by PMDL prescribed model does not predict the right value for y_i. This bit string can be coded using $\log \binom{m}{F}$, where F is the number of errors made by the models prescribed by PMDL. Minimizing this quantity is equivalent to minimizing F. Therefore we simply count the mistakes each model class $\mathcal{M}_k$ makes when we predict each example x_t using the ML model, $\hat{M}_k(x^{t-1}) \in \mathcal{M}_k$, of its predecessors.[3] Let $\hat{y}_i$ be the prediction of $\hat{M}_k(x^{i-1})$ for y_i. The new evaluation criterion is given by:

$$F_{\mathcal{M}_k}(S(m)) = \sum_{i=0}^{m} |\hat{y}_i - y_i| \tag{6.4}$$

We select the model class $\mathcal{M}_k$ that minimizes Equation 6.4. Within that model class we select the ML of the complete data. It can be shown that using (6.4) is in fact equivalent[4] using (6.3), see [Grü98] Section 2.2.

6.4 Analyzing PMDL

In this section we analyze the experimental results that we obtained for PMDL. The algorithm that was implemented is basically the same as the one introduced in Section A.5. However, instead of the loop over precision, we now have a loop over the examples. We start by predicting the second example from the ML model of the first example. Next, we predict the third example from the ML model of the first and the second example, etc. For the model classes that contain models that are more complex than the simplest consistent model, we use the prediction of the simplest consistent model. This is justified because there is always a model in the more complex model class that performs as well as the consistent model. We count

[3] Another way to work around this problem would be to associate a distribution $P_M(\cdot)$ with each model $M \in \mathcal{M}_k$, such that $P_M((x_i, M(x_i))) = \hat{\epsilon}(\mathcal{M}_k, S^{1\ldots i-1})$. We let $\hat{\epsilon}(\mathcal{M}_k, S^{1\ldots i-1})$ denote the fraction of the first $i-1$ examples in the sample S that was correctly predicted by the models selected by PMDL. However, we have not been able to compare this solution with the one we used.

[4] We can find a probability distribution P such that when we use (6.3) with P we will select the same models as when using (6.4).

the number of errors the ML models of each model class made, and we record for each sample size which model class yielded the fewest prediction errors.

6.4.1 The Noiseless Situation

When we look at the behavior of PMDL on the interval function selection problem, we see that PMDL needs a certain minimum number of examples to estimate the right number of switches. The number of switches of models in the model class selected by PMDL as a function of the number m of examples available to the PMDL model class selection algorithm is well approximated by $\hat{d}(m)$, as given by the following formula:

$$\hat{d}(m) = d(1 - (1 - 1/d)^{m-m_0}), \tag{6.5}$$

where d is the number of switches in the data generating function. See Figure 6.1 for a plot of both the approximation and the actual behavior. Equation 6.5 can partially be (partially) explained as follows: The probability that a particular example is not in some interval I is $\frac{d-1}{d}$. Here, we assume that the examples are drawn according to a uniform distribution as was the case in our experiments. The probability that there is no example in some particular interval I is $(\frac{d-1}{d})^m$, so the probability that there is at least one example in I is $1 - (\frac{d-1}{d})^m$. Now we define $Y = \sum_{i=1}^{d} X_i$, where each random variable $X_i \in \{0, 1\}$ denotes whether or not there are some examples in the i^{th} interval. So Y denotes how many intervals have one or more examples in it. Because the expectation of the sum equals the sum of the expectations, $E(Y) = \sum_{i=1}^{d} E(X_i)$, the expectation of Y is given by $d(1 - (1 - 1/d)^m)$.

Note the shift to the right of d_0 with m_0. An experimentally good approximation of m_0 is given by $(\log d - \log (d-1))^{-1}$. Note that m_0 gives us the value for m for which we expect to have an example in the sample from at least half of the intervals in the target function. For example: if $d = 100$ then $m_0 \approx 69$. If $m < m_0$ we have that, with probability greater than $1/2$, if we draw an example it will not be in an interval that is already represented by one of the m examples we have seen so far. So it is unlikely that good predictions can be made and thus that some ML model with more than zero parameters out-performs the zero-hypothesis.

However, it remained unclear exactly how the expression for $\hat{d}(\cdot)$ represents the behavior of PMDL on this problem domain for $m \geq m_0$.

6.4.2 Dealing with Noise

Here, we will take a look at the situation where the sample is corrupted by some noise, that is to say, each example in the sample has a probability η of being labeled incorrectly. PMDL will perform a small amount of overcoding when few examples are given to the model class selection algorithm. This is due to the fact that when there are relatively few examples, there is, for a large fraction of the examples when assuming uniform distribution over the example space, relatively small probability that the next example is located in one of the regions where the ML model of the preceding data does not model the target function correctly. Thus when there are relatively few examples, there is a relative small probability that the ML model of the preceding data makes an incorrect prediction.

If we look at the rate of overfitting, we see that PMDL does not select the class $\mathcal{M}_{d_0}$ which enables us to code the sample consistently. Rather, it selects model classes that overfit the data less extremely. See Figure 6.3 on page 59 for an illustration. We can explain the overfitting of PMDL intuitively in two stages; see Figure 6.3 for an illustration.

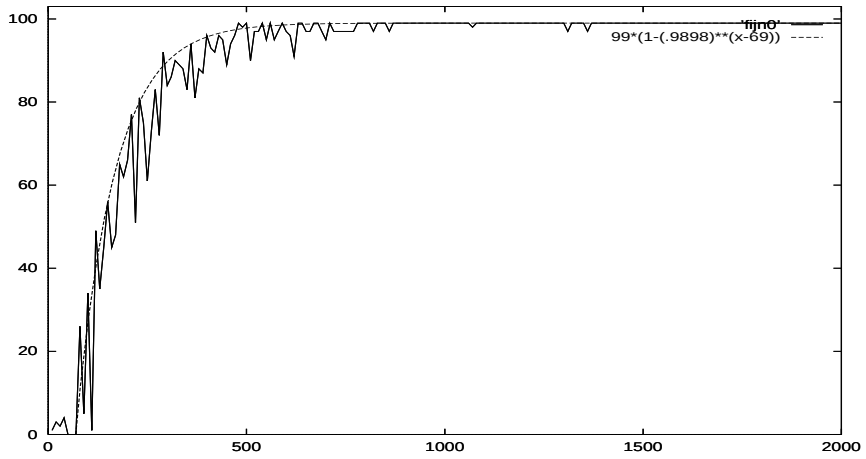

Figure 6.1: Plot of the number of parameters selected by PMDL against sample size. We used a noiseless sample and a target function of 100 intervals, where the x-values are drawn according to a uniform distribution. The smooth function is $\hat{d}(100)$.

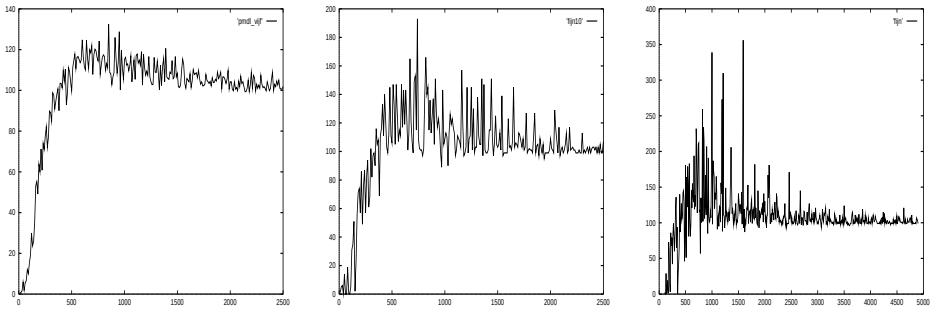

Figure 6.2: Plots of number of parameters selected by PMDL against sample size. We used noise rates 0.05, 0.1 and 0.2, from left to right. In both cases the target function had 100 intervals.

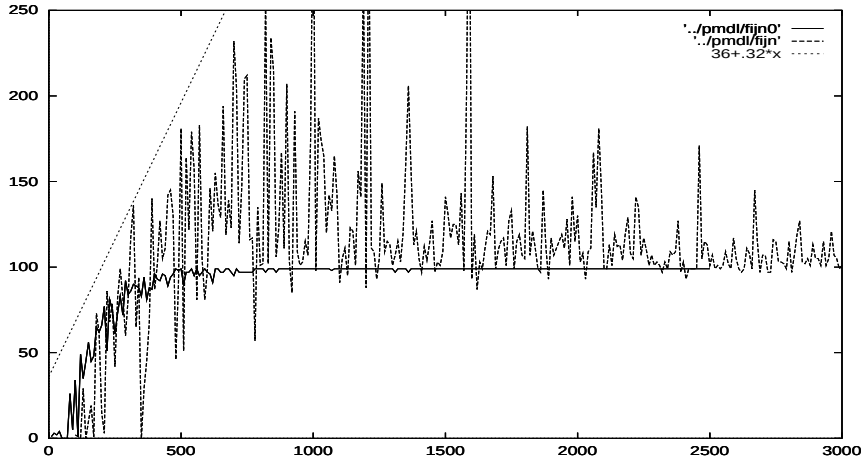

Figure 6.3: Plot of the number of parameters selected by PMDL against sample size. We used a noiseless sample for the smooth curve; 20% noise was used in the instable curve and the straight line represents the expected number of switches in the data, d_0. In all cases a target function of 100 intervals was used.

Why can models which are too complex be preferable?

Imagine the situation in which only relatively few examples of the data generating function are available. Assume that the number of switches, d_0, in the consistent model for the sample is greater than d. Let $\hat{h}_d$ be the hypothesis that minimizes the error on the sample among the hypotheses with d switches, we call $\hat{h}_d$ optimal in the class of hypotheses with d switches. If we have only few examples available, we have relatively high probability that the $\hat{h}_d$ does not represent all the switches in the data generating function. It is quite likely that this hypothesis models some switches that originated from some noisy examples in the data. However, if we use a hypothesis with more than d switches, say d', we are sure that some switches are modeled that are not present in the data generating system. Second, we expect to model more switches of the data generating function when d' switches are modeled. This is easily seen from the fact that if we model all d_0 switches in the data then *all* the switches of the data generating function that are represented in the data are modeled. The more switches we model, the more likely it is that we model more switches in data that represent switches in the target function. If relatively few examples are present in the data, we have a smaller probability of making one or more incorrect predictions with models which are too complex, than when we have of making one or more incorrect predictions when using a model with fewer switches.

Bad prediction by models which are too complex

As we use bigger and bigger samples, the cost of modeling some noisy switches; that is switches that are not present in the data generating function, will increase. This is due to the fact that when we have modeled some noisy interval, the probability that there is some example in the rest of the sample that falls within this interval and is **not** corrupted with noise increases with m. The latter event coincides with the event that an incorrect prediction is made. Therefore, as the sample size m increases, the number of intervals in available in the model class that contains the best predicting model tends to approach the correct value d.

The plots of Figure 6.2 show a rather surprising effect: the number of examples that yields the greatest overfitting seems to be independent of the noise rate. This suggests that there is some fixed value m^* for which the overfitting tends to disappear as more and more samples are drawn in noisy intervals and too complex hypotheses make bigger and bigger prediction errors. Unfortunately, there was no time to investigate how the value of m^* depends on the target function any further within this project.

6.5 PMDL and Cross Validation

Apart from MDL and GRM, Cross Validation is the third model selection method that is treated in [KMNR97]. In this section we explain the strong resemblance in the performance of Cross Validation and Predictive MDL on the interval function selection problem.

Before proceeding we want to stress here that what is written in this section is applicable to all model selection problems where the model class consists of indicator functions, not just the interval function selection problem.

We will first briefly discuss the Cross Validation model selection method.

6.5.1 Cross Validation

When using Cross Validation (CV), the sample is divided into two disjoint sets, a training set and a testing set. By means of Maximum Likelihood we find the optimal model in each model class for the training set. The second step is to measure the error each optimal model makes on the test set. The model which yields the lowest test error is selected by CV.

Let γ be the fraction of the examples that is present in the training set. Let $\hat{M}_i(S(m))$ be the Maximum Likelihood model for sample $S(m)$ of length m within model class $\mathcal{M}_i$. Now CV will select model M such that:

$$M = \arg\min_{\hat{M}_i(x^{\gamma m})}\{\hat{\epsilon}(\hat{M}_i(x^{\gamma m}), x_{\gamma m+1}, \ldots, x_m)\},$$

where the training error is defined as in Definition 3.3 on page 22 and $x^{\gamma m}$ are the first γm examples of the sample.

We could look at CV as an instance of the Two-Part MDL principle in which we only want to compress the second part of the data, $x_{\gamma m+1}, .., x_m$. However, we restrict the model class to ML models of the first part of the data, $x^{\gamma m}$, and do not count any penalty for complex models. Intuitively CV is appealing as we first search for models that describe the first part of the data very well, the ML models for $x^{\gamma m}$. Second, from those models we pick the model that gives the best compression of, or equivalently: lowest error on, the second part of the data, given the model. We hope that in this way we choose a model that yields a short description of the complete data. The idea is that models that overfit or underfit the first part of the data will yield bad performance on the second part of the data and thus those models will not be selected. However, it is in general unclear what value for γ should be used to obtain optimal results.

6.5.2 CV as an Approximation of PMDL

In this section we compare PMDL and CV. Note that constructing the PMDL code is quite like the process of finding the best model in terms of CV. When applying PMDL we pick the model $M_i = ML(\mathcal{M}_i|x^n)$ for some sequence $x^n, n < m$, taken

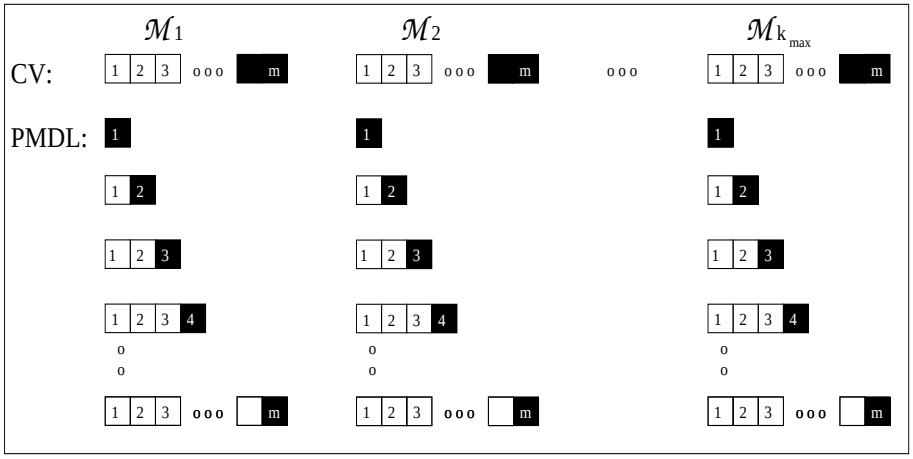

Figure 6.4: The white boxes are used to predict the black boxes. For each model class $\mathcal{M}_1$ up to $\mathcal{M}_t$ Maximum Likelihood models are computed. PMDL needs to compute m, the sample size, ML models. CV settles for only one ML model per model class.

from a sample of length m, and predict the next example x_{n+1} with it. See Figure 6.4 for an illustration of the difference between PMDL and CV. Both CV and PMDL have the same underlying strategy: pick the ML models for some part of the data sequence and prefer the one that makes the smallest error on some other part of the data.

The main difference between PMDL and CV is that PMDL does not use one model from each model class $\mathcal{M}_t$ under consideration, but may use $m-1$ different models from each class. So PMDL says something about model classes instead of individual models. If we want to use an individual model, for example for prediction, we should pick the ML model in the model class advocated by PMDL, as if we were continuing the predictive coding on new data.

A related difference is that CV uses only a fraction γ of the observed data to select the ML model of each model class $\mathcal{M}_t$. So, there is always a fraction of the data that is not used to select the candidate models, but this fraction is used merely to evaluate those candidate models. However, the PMDL method exploits all examples (except for the last one) for selecting candidate models. PMDL also uses all examples (except for the first one) to evaluate the candidate models. The number of times an example x_i is engaged in selecting candidate models for PMDL is given by $m-i$.

It is clear that if the beginning of the data sequence consists of very a-typical examples then the ML model of the target model class will differ substantially from the ML model in that class over all data. It is therefore likely that the ML model of the target model class will make a substantial number of incorrect predictions on the preceding examples. So if we have 'bad luck' with ordering the data and the data sequence starts with a-typical examples, PMDL is less likely to select the target model class.

If we look at CV, we see that it suffers from a similar defect as PMDL. If the training (or test) set for CV contains many a-typical examples, the ML models that are selected on the basis of this training set are unlikely to model the examples in the test set very well. The plots of Figure 6.5 confirm that the greater we pick $\gamma \geq 1/2$ for CV, the less stable the behavior of CV becomes. This is due to the partition dependence.

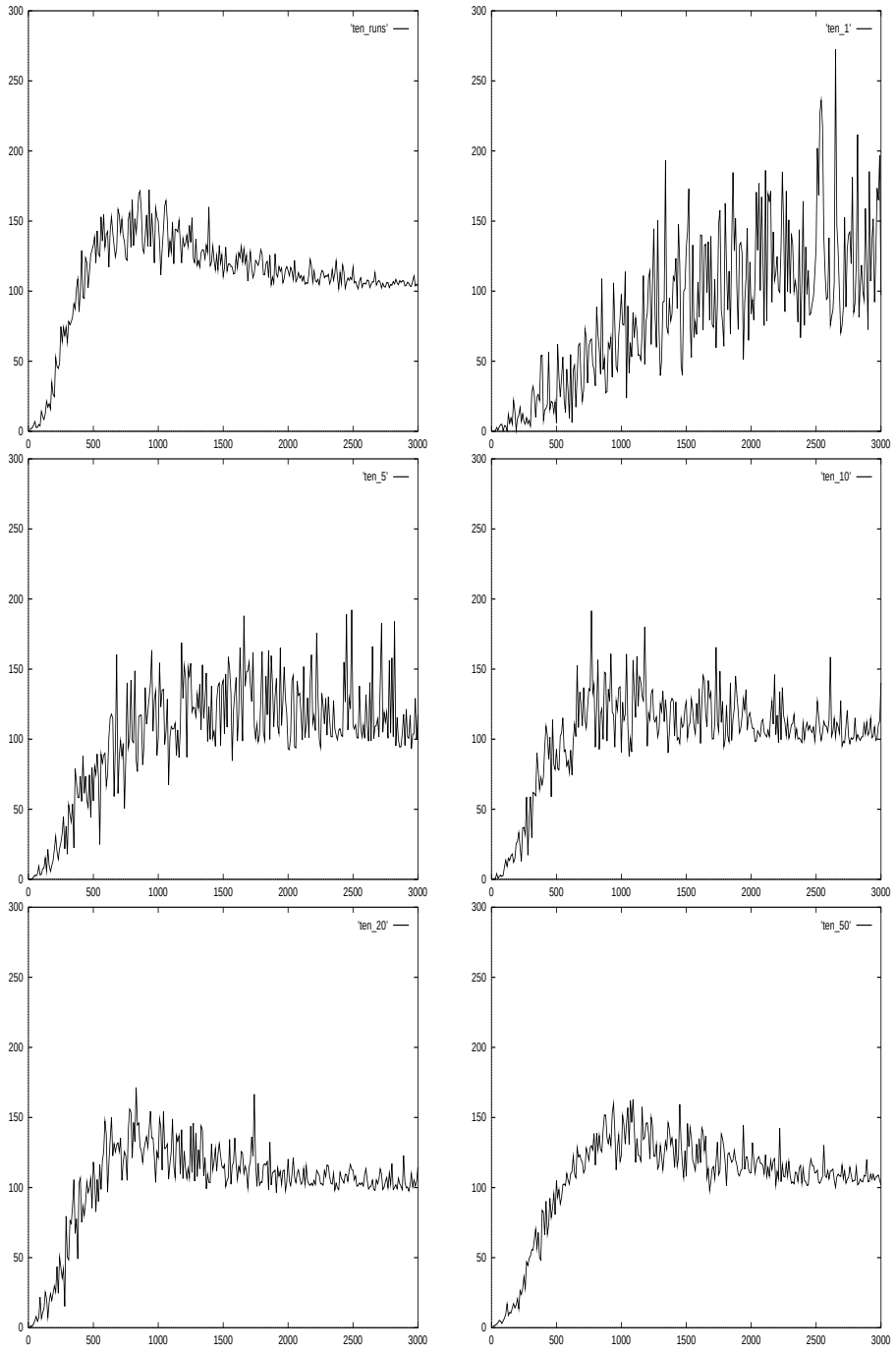

Figure 6.5: Plot of the selected number of parameters against sample size, each data point represents the average over ten experiments. A target function of 100 intervals was used and 20% noise was added. The upper left plot shows performance of PMDL. From left to right and top to bottom the other plots display performance of CV using $\gamma = 0.99, \gamma = 0.95, \gamma = 0.9, \gamma = 0.8$ and $\gamma = 0.5$.

We conclude that CV can be seen as an approximation to PMDL, where the data is viewed as consisting of two segments. The price we pay for using the more subtle, and non-parametric, PMDL instead of CV is the longer running time of PMDL. Let t be the number of model classes. Using PMDL we would now have to estimate $t(m-1)$ maximum likelihood models over an average of $m/2$ examples. Using CV one only has to estimate t maximum likelihood models over $\gamma m > m/2$ examples.

Chapter 7

More Experiments for Two-Part MDL

In this chapter we present the second series of experiments that we conducted. First, in Section 7.1, we discuss why we conducted the experiments. Next, in Section 7.2, we discuss the problem domain. In Section 7.3 the methods used are discussed. Finally, in Section 7.4, we discuss the results of the experiments.

7.1 Motivation for More Experiments

In the previous chapters we have seen that the MDL Two-Part principle displayed better performance, in the sense of model class estimation, when no use was made of sample-specific information to restrict the model classes. However, it remained unclear whether this was due to the *restrictions* on the different model classes or due to the fact that it was *sample-specific information* that was used to restrict the model classes. Furthermore, it remained unclear whether or not the overcoding is restricted to the interval function selection problem. To answer these questions we have conducted a second series of experiments in a different problem domain.

In these experiments the problem of data clustering is used. Clustering is a well known problem in Artificial Intelligence that will be discussed in the next section.

7.2 Problem Setting

In this section we describe the second type of experiments we have conducted to provide answers to the questions that were posed in the previous section. We start by defining the problem of data clustering. Using this definition we discuss what is the inference task for the MDL model selection algorithm.

7.2.1 Clustering

First we introduce the concept of clustering:

Definition 7.1 *A* **clustering** $\mathcal{C}_{\sqcup} = \{C_1, C_2, ..., C_t\}$ *of an unlabeled sample* $S(m)$ *is a partition of* S *in* t *disjoint subsets or* **clusters**.

The first problem of clustering is to find the optimal number of clusters. The second task is to find, for a fixed number of clusters, t, the optimal partition of the data into t subsets. Now we will informally specify what the optimal number of clusters is and what an optimal partition of the data into t subsets is.

When clustering the data into say t clusters the aim is to divide the data into t subsets such that all data examples that display 'natural similarity' are put in the same cluster, where unrelated objects will belong to different clusters. The other, and higher level, goal to find the optimal number of clusters is fulfilled for that number of clusters which allows the 'best' trade-off between similarities within each cluster and differences between different clusters. This trade-off leads to a choice for a clustering that is somewhere between the two trivial clusterings C_1 and C_m, where m is again the number of examples that was given to the model selection algorithm :

- C_1**:** All examples are clustered in one cluster; the clustering gives very little information about the examples. If all examples belong to the same cluster, the cluster can be seen as a statement like: "All data is concentrated around some point P, with an (empirically determined) variance of σ." Therefore, it does not contain much information on all particular data objects.

- C_m**:** Each example has its own cluster; the clustering does not give any information on which examples display similarity.

In general, when we want to cluster m data examples into t clusters we cannot evaluate all possible clusterings, as there are simply too many of them. Take for example[1] $m = 19$ and $t = 3$. In this case, there are approximately $1.93 \cdot 10^8$ possible clusterings[2].

As there are so many possible clusterings, we give up an exhaustive search for the best clustering and try to find a good one by means of:

1. An efficient search algorithm.

2. A function that evaluates the quality of candidate clusterings.

We can divide clustering algorithms into two main categories. The first we call Hierarchical algorithms, the second we call Partitional algorithms. A Hierarchical algorithm starts by clustering the data such that each example gets its own cluster. The next step is then to merge the two clusters C_i and C_j that are the most similar according to some distance measure $D(\cdot, \cdot)$. The algorithm keeps merging clusters until it ends up with just one cluster, or we let it stop as soon as it reaches t clusters. The result is a nested structure of clusters.

Partitional clustering algorithms are intended to find a sub-optimal clustering of m examples into t clusters. The strategy of the algorithms is to iteratively optimize the evaluation function, where the least squared error measure is a common choice for the evaluation function. Often one starts by generating random cluster centers. The squared error evaluation function for one cluster C_k is given by:

$$\varepsilon_k^2 = \sum_{x_i \in C_k} (x_i - \mu_k)^2 \tag{7.1}$$

Where μ_k is the center of the k^{th} cluster. By using the squared error evaluation measure the clusters have the form of a *hypersphere*[3], as we assign each data-point to the cluster with the closest center. The evaluation measure for the total clustering is given by the sum over the measures for the individual clusters:

$$E_t^2 = \sum_{k=1}^{t} \varepsilon_k^2 \tag{7.2}$$

[1] This example and the setup for introducing clustering were taken from [BD89]

[2] The number of clusterings is actually given by $\{{}^m_t\}$, a Stirling number of 'the second kind', denoting the number of ways we can divide a set of m things into t non-empty subsets, see [GKP89] page 243.

[3] A hypersphere is an n-dimensional sphere. It is defined using its center μ and some radius r: $\{x : |x - \mu| \leq r\}$.

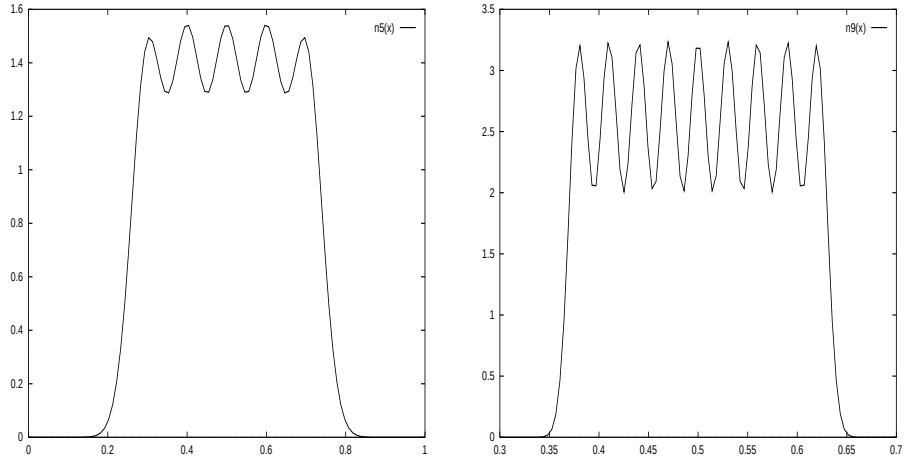

Figure 7.1: Plots of two target functions, density functions that are mixtures of five and nine normal distributions, respectively. The variance of all normal distributions is equal and given by 1/320 and 1/5120, respectively. The distance δ between the distributions is 1/10 and 1/30, respectively.

7.2.2 The Data Source and Inference Task

In the experiments we have conducted, we used one-dimensional data generated by a mixture of d normal distributions. All d normal distributions have the same variance[4] $\sigma^2 = (10 \cdot 2^d)^{-1}$. The distance between the means μ_i and μ_{i+1} of two normal distributions, $\mu_{i+1} - \mu_i = \delta$, is fixed for all $i \in \{1, ..., d-1\}$. The mean of the means $\mu_1, \mu_2, ..., \mu_d$ is fixed at 1/2, so $\frac{1}{d}\sum_{i=1}^{d} \mu_i = 1/2$. Each normal distribution is weighted equally. Summarizing, the examples are drawn according to the density function:

$$f(x) = \sum_{i=1}^{d} \frac{1}{d\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu_i)^2}{\sigma^2}} = \frac{1}{d}\sum_{i=1}^{d} \frac{1}{\sqrt{2\pi(10 \cdot 2^d)^{-1}}} e^{-(x-\mu_i)^2(10 \cdot 2^d)} \qquad (7.3)$$

Why the restriction that the variances of all normal distributions are equal was introduced is made clear in the next section.

The task of the inference algorithm is to predict the number of normal distributions d in the data generating distribution, given a sample $S(m)$. The second task is to find the means and variances of the d normal distributions. In the experiments we have used, the Two-Part MDL principle in combination with four different model classes. These model classes differ in how much knowledge of the target function is present. We will discuss the four model classes in the next section.

7.3 Method

In this section we describe the methods that were used to implement a solution to the problem that was outlined in the previous section. We divide this solution in two parts. First we discuss the **FORGY**[BD89] algorithm that we have used to generate clusterings. After that, we discuss the different model classes and corresponding model class evaluation functions we have implemented.

[4] This value was chosen because at this value it is highly improbable that examples outside $[0, 1]$ are drawn.

7.3.1 Clustering

For the clustering we decided to use the classical[5] **FORGY** algorithm, which is a Partitional algorithm. The input for the **FORGY** algorithm consists of the number of clusters t that should be generated and the data that should be clustered. The **FORGY** algorithm performs the following steps to output a partition of the data into t subsets:

1. **Initialize the t cluster centers.** This is done by assigning t different randomly chosen examples as cluster centers.
2. **Label each example** with the index of the cluster with the closest cluster center. Here, we used the Euclidean distance as distance measure. As we have restricted the admissible target models to mixtures of normal distributions with equal variance, it is a good idea to use Euclidean distance. Below we explain why.
3. **Compute new cluster centers.** The mean of all examples belonging to the cluster is computed and used as the new cluster center.
4. **Perform step 2 and 3** until the clustering remains the same, i.e., has converged, or until some fixed number of iterations is reached.

The output of the **FORGY** algorithm is a set of t cluster centers, $\{\mu_1, \mu_2, ..., \mu_t\}$. This set is fed, together with the data, to the MDL Two-Part model selection component. From these two ingredients a clustering of the data is generated as follows: Assign each example x_i to the cluster C_{t_j}, if μ_j is the closest cluster center for x_i.

The MDL component, for each clustering, computes the code length of the data using the clustering and the description length of the clustering itself. The summation of these two description lengths gives us the total code length of the data using a model from one of the four model classes we will describe in the next subsection. We let the **FORGY** algorithm compute one clustering for each number of clusters t, where t can take values from 1 up to some value $t_{max} \leq m$. Obviously not all the models in the model class are evaluated, only those t_{max} models with t_{max} or fewer clusters that were selected by the **FORGY** algorithm are evaluated.

The least squared error method of cluster optimization in the **FORGY** algorithm induces using intervals as clusters in the case of one-dimensional data. In this case an example belongs to the cluster with the closest cluster center.

7.3.2 Model Evaluation

To evaluate models according to the Two-Part MDL principle, we need to:

1. code the model / clustering, or at least compute the code length for the model.
2. compute the code length of the data using a model.

We start by describing the second task.

Coding Data with a Clustering

To code the data with a model we have to use the output of **FORGY** to compute code lengths for the data. We reinterpreted the output of **FORGY** as an estimation of normal distributions on the data. Each cluster represents a normal distribution, and the total clustering C_t is interpreted as the mixture of t normal distributions.

[5] See [BD89].

As pointed out in [Grü98] Chapter 1, we can derive code lengths from density functions much in the way we derived code lengths from probability distributions. In this thesis, we only consider continuous density functions.

Let $f(\cdot)$ be a density function on the domain X. Let X' be a finite subset of X. There exists a prefix-free coding scheme $C : X'^* \to D^*$ that maps sequences of objects from X' to sequences of symbols of some alphabet D. The code length $L_C(x)$ that C assigns to each object $x \in X'$ is (the ceiling of) $-\log f(x) + K$, where K is some constant independent of x:

$$L_C(x) = -\log f(x) + K, x \in X' \tag{7.4}$$

It can be shown that under some mild conditions on the model class (which apply here), (7.4) holds for all models in $\mathcal{M}$ (where the same K can be chosen for all models). We therefore ignore K in the computation of the code lengths below. See [Ris87] for more details on the derivation of code lengths from density functions.

As we are using a mixture of t normal distributions, with weight factors $w_1, ..w_t$, the code length for an example $x \in [0, 1]$ given in Equation 7.4 becomes[6]:

$$L_C(x) = -\log \sum_{i=1}^{t} f_i(x) w_i = -\log \sum_{i=1}^{t} \frac{w_i}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu_i)^2}{\sigma^2}}, \tag{7.5}$$

with: $\forall_i : w_i > 0$ and $\sum_{i=1}^{t} w_i = 1$.

So the total code length for a sample x^m of m examples gets, as the examples are supposed to be independently identically distributed (i.i.d.):

$$L_C(x^m) = -\log f(x^m) = -\log \prod_{i=1}^{m} f(x_i) = -\sum_{i=1}^{m} \log f(x_i), \tag{7.6}$$

where f is again the density function over the sample space. Now we can use the expressions in Equation 7.5 to make Equation 7.6 more detailed. We arrive at

$$L_C(x^m) = -\sum_{j=1}^{m} \log \sum_{i=1}^{t} \frac{w_i}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_j-\mu_i)^2}{\sigma^2}} \tag{7.7}$$

as a final expression for the code length of a sample using a mixture of normal distributions.

Coding Models with Respect to Four Different Model Classes

To make things a little bit less complex, we have made some restrictions on the data generating function. These restrictions allows us to restrict our model classes too, without taking the risk that the models that we have left will not be able to fit the data reasonably well. First of all recall how the data generating functions are restricted as described in Subsection 7.2.2. Now we describe the model classes that were used in different implementations of the MDL model selection component. We have used four different model classes, yielding four different coding schemes and four different model evaluation functions. We list them here in such a way that we restrict the model class more and more:

[6] Actually, this coding scheme is a little redundant. We know that the domain of our data is limited to $[0, 1]$. However the normal distribution assigns mass to an infinite domain. To fix this redundancy we should compensate for the mass assigned outside $[0, 1]$ by multiplying the density function by some constant. Since all models we consider hardly put any mass outside [0,1], we can safely ignore the redundancy.

- **Restriction 1: Mixtures of Normal Distributions**

  We restrict the model class to those mixtures of normal distributions such that the mean of all distributions is distinct and each distribution has the same weight $1/t$, where t is the number of distributions that is mixed. We denote this model class by

  $$\mathcal{M}^{r1} = \bigcup_{1 \leq i \leq m} \mathcal{M}_i^{r1},$$

  where $\mathcal{M}_i^{r1} = \{f_i(\cdot)\}$ and $f_i(\cdot)$ is a mixture of i normal distributions with the restrictions given above.

  Now we consider the number of bits we need to code such a description of a model $M \in \mathcal{M}_t^{r1} \subset \mathcal{M}^{r1}$. Suppose that we code parameters with p bits precision. We first need to specify the numbers t and p; we can code them prefix-free in $L^*(p) + L^*(t)$ bits, where $L^*(\cdot)$ is defined as on page 19. Now we need to code the parameter values. As we have used one-dimensional data, we can completely determine each normal distribution by specifying its mean μ and the variance σ^2. We can use the idea of the MDL_{pss} coding scheme (see page 24) to code the values of the means, as we know that all mean values are distinct. This means we need $\log \binom{2^p}{t} \approx 2^p \mathcal{H}(t/2^p)$ bits.[7] In order to know which variance belongs to which mean, we code the variances in the order of increasing corresponding means. For each variance we need p bits. Therefore, we need tp bits to code all the variances. The total code length for a model M now becomes:

  $$L_{\mathcal{M}^{r1}}(M) = tp + 2^p \mathcal{H}(t/2^p) + L^*(p) + L^*(t) \tag{7.8}$$

- **Restriction 2: Variance is Known**

  Suppose that we know that the variance of the data generating distributions is given by $\sigma^2 = (10 \cdot 2^t)^{-1}$. The inference task that is left is to estimate the number of normal distributions that generated the data and to locate them. We define the resulting model class $\mathcal{M}^{r2}$ as follows:

  $$\mathcal{M}^{r2} = \bigcup_{1 \leq i \leq m} \mathcal{M}_i^{r2}$$

  $$\mathcal{M}_i^{r2} = \{M \in \mathcal{M}_i^{r1} : \text{ for } j \in \{1, \ldots, t\} \sigma_j^2 = (10 \cdot 2^i)^{-1})\}$$

  Now we can code each model $M \in \mathcal{M}^{r2}$ in much the same way as we coded models from $\mathcal{M}^{r1}$, except that we do not have to code the variances. The resulting code length for models from $\mathcal{M}^{r2}$ is:

  $$L_{\mathcal{M}^{r2}}(M) = 2^p \mathcal{H}(t/2^p) + L^*(p) + L^*(t) \tag{7.9}$$

- **Restriction 3a: Using Examples as Means**

  The model class $\mathcal{M}^{r3a}$ is a subset of $\mathcal{M}^{r2}$. The values allowed for the means are those of examples from the sample, $S(m)$.

  $$\mathcal{M}^{r3a} = \{M \in \mathcal{M}_t^{r2} : \forall_{k \in \{0, \ldots, t\}} \exists_{x_i \in S(m)} \mu_k = x_i\}$$

  In this situation we only have to specify the number of distributions, t, that are being mixed and the subset of examples that are used as values for the means. As there are $\binom{m}{t}$ subsets with t elements of a set with m elements, we have:

  $$L_{\mathcal{M}^{r3a}}(M) = m\mathcal{H}(t/m) + L^*(t) \tag{7.10}$$

[7] See page 24 for this approximation.

- **Restriction 3b: Using Constant Distance Between Means**

  The last model class we consider, $\mathcal{M}^{r3b}$, is again a subset of $\mathcal{M}^{r2}$. The distance between the means of two consecutive distributions in a mixture is fixed at δ and second, the mean of the means is fixed at $1/2$. In this situation we only have to specify the number of distributions, t, that are being mixed and the constant distance, δ, between the means of neighboring distributions. The value of δ is coded using p bits. We have:

$$L_{\mathcal{M}^{r3b}}(M) = p + L^*(p) + L^*(t) \tag{7.11}$$

Often the sender-receiver model[8] is a very clarifying point of view when we are dealing with coding and decoding. However, it does not apply when we are using model class $\mathcal{M}^{r3a}$. In this model class we use the data to *describe*, or code, the model with which we want to code the data. Notice that we cannot use the model to *send* the data, as the data must be known to decode the model. This is strictly speaking an incorrect way to apply the MDL principle. However, we included it nevertheless since:

- It is, to some extent, analogue to the use of x-values by the $KMDL$ coding scheme in the interval function selection problem, as introduced in Chapter 3.

- It does seem to make sense as an estimation procedure, as experiments showed good performance.

For a description of the implemented algorithm we refer the reader to Appendix A, Section A.6.

7.4 Results of Experiments

In this section we review the performance yielded by the use of the four different model classes. If we write $\mathcal{M}^{r3}$, we mean both $\mathcal{M}^{r3a}$ and $\mathcal{M}^{r3b}$. The results are summarized in Figure 7.2 and 7.3.

Overfitting

We see that for a target distribution consisting of the mixture of five normal distributions, MDL using the model class $\mathcal{M}^{r3}$ tends to select models that consist of more distributions than the target. The use of the other two model classes does not lead to overcoding.

Rate of Convergence

We see that $\mathcal{M}^{r2}$ and $\mathcal{M}^{r3}$ yield about the same rate of convergence to the target complexity of modeling, although the latter is somewhat faster. When we use model class $\mathcal{M}^{r1}$ the convergence is much slower, due to a higher complexity term as compared to $\mathcal{M}^{r2}$ and $\mathcal{M}^{r3}$. See Figure 7.2 on page 73 for plots of the model selection behavior of Two-Part MDL using the three different model classes.

[8] See Section 2.5

Conclusion

Recall that we conducted the experiments described in this chapter to answer two questions: First, we wanted to know whether we needed sample-specific information to shrink the model class in order to have MDL perform overfitting. Second, we wanted to know whether MDL would also perform overfitting for small model classes when models other than interval functions were used. Our experimental results seem to show that:

First, the information used to shrink the model classes does not have to be sample-dependent to lead to overfitting, because MDL performed overfitting for both model class $\mathcal{M}^{r3a}$ (sample-dependent) and $\mathcal{M}^{r3b}$ (sample-independent). Figure 7.2 on page 74 illustrates that, when using a data generating function which consists of five normal distributions, for sample sizes $m > 50$ we can hardly call the performance 'overfitting' anymore.

We explain the overfitting of model classes $\mathcal{M}^{r3a}$ and $\mathcal{M}^{r3b}$ as follows: For (very) small samples we can find a mixture (model) that puts mass only on very specific and small areas, such that most of the examples are located in those areas. Such models assign relatively little mass to those areas where no data is observed. Therefore the data can be coded in relatively few bits using those models. The models in model class $\mathcal{M}^{r3a}$ and $\mathcal{M}^{r3b}$ can be coded in very few bits, so the total code length for the data is small. However, as we used relatively small values for c_{max} (the maximal number of normal distributions that were mixed) the overcoding stopped for (very) small sample sizes. For greater values of c_max overcoding may sill be performed for greater sample sizes. If we use a model that puts mass only on specific and small areas and there is data D' which is not in those areas, then D' will have to be coded using many bits and a model which consists of fewer distributions will be selected.

Second, the overfitting for small model classes is not restricted to modeling interval functions. The last claim can be made because the experiments we described in this chapter use mixtures of normal distributions as models.

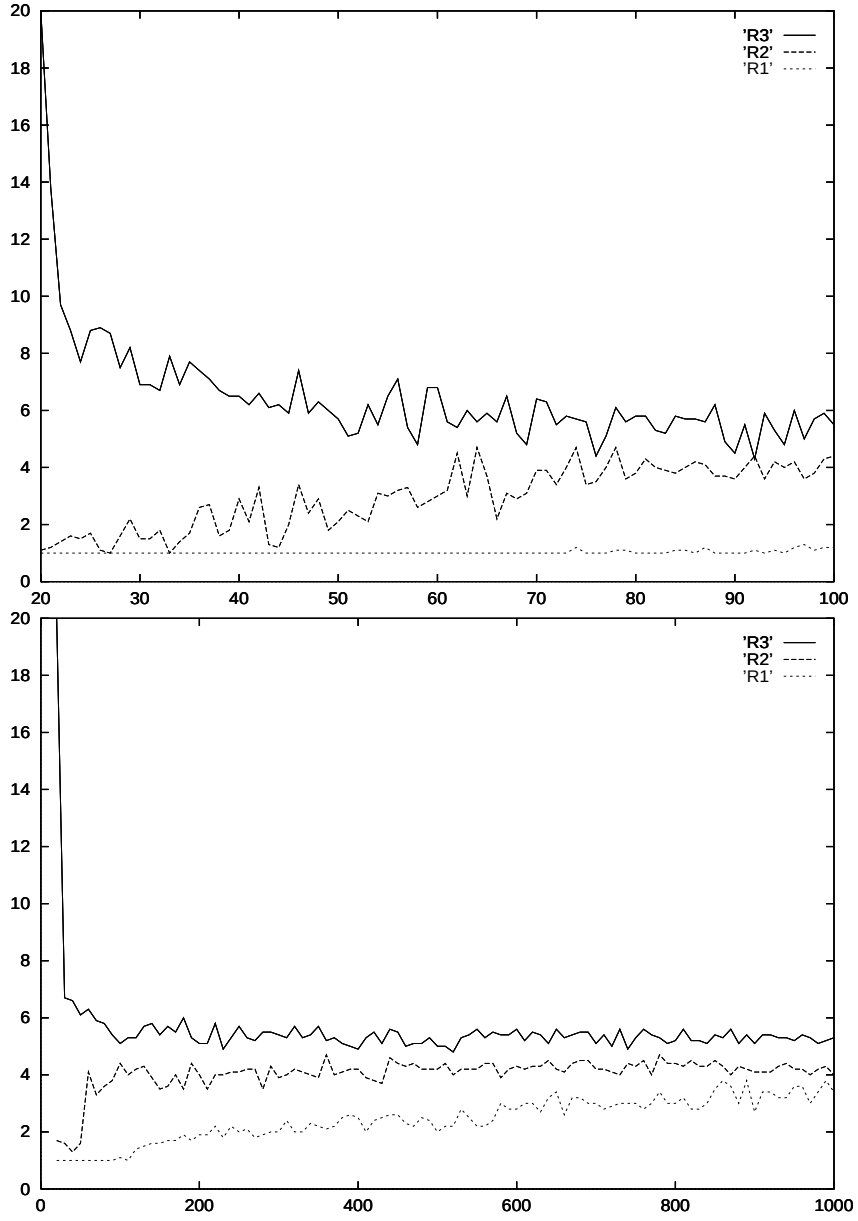

Figure 7.2: Plots of the number of distributions selected by Two-Part MDL against sample size. Model classes $\mathcal{M}^{r1}$, $\mathcal{M}^{r2}$ and $\mathcal{M}^{r3a}$ were used. The upper plot is a close-up of the lower one. Each data point represents the average of ten experiments. The results for model class $\mathcal{M}^{r3b}$ are not plotted because they are very similar to those of $\mathcal{M}^{r3a}$, as is illustrated for small m in Figure 7.3.

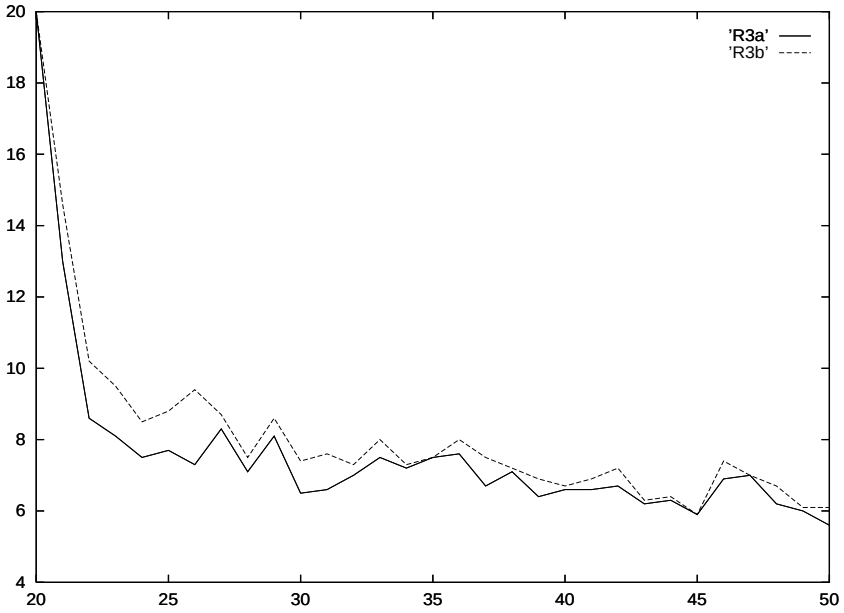

Figure 7.3: Plots of the number of distributions selected by Two-Part MDL against sample size, using model classes $\mathcal{M}^{r3a}$ and $\mathcal{M}^{r3b}$. Each data point represents the average of ten experiments. If larger samples are used, the difference between the performance of model classes $\mathcal{M}^{r3a}$ and $\mathcal{M}^{r3b}$ gets less and less.

Chapter 8

Discussion of Results and Conclusions

In this chapter we will look back at the different inference algorithms we have treated in the preceding chapters and compare them. An overview of the performance of the model selection methods on the first problem domain is given by Figure 8.1 on page 79.

8.1 Discussion of Two-Part MDL Coding Schemes

Both Two-Part MDL coding schemes that were introduced in Section 3.2, MDL_{pss} and MDL_{tp}, do not display the overcoding behavior of $KMDL$. However, we see that MDL_{tp} needs many examples before it will approach the right number of switches, whereas MDL_{pss} has a higher rate of convergence than $KMDL$. This confirms our conjecture that the coding scheme for Two-Part MDL should be sample-independent. The first two questions we stated in Chapter 1 can now be answered. We have seen that there is no *formal* reason to reject the $KMDL$ coding scheme. On the other hand we have seen that the performance of Two-Part MDL, in terms of estimating the number of parameters in the data generating function, can be improved by using an efficient and sample-independent coding scheme.

We observed that MDL_{pss} does not display the extreme overfitting that $KMDL$ displays and MDL_{pss} converges considerably faster to the correct model complexity. However, the actual *parameter estimation* of MDL_{pss} is of such poor quality that $KMDL$ outperforms MDL_{pss} in terms of *generalization error*. We noted that using Two-Part MDL just to select the *number* of parameters and ML to estimate the parameter *values*.

8.2 Size of the Model Class

In this section we examine the effect of shrinking the model class on the performance of model selection methods.

8.2.1 Two-Part MDL

We start with a (very) brief review of Two-Part MDL. After that we consider the effect of using a small model class for the interval function selection problem. Finally we present a more general discussion on the influence of the model class size on the performance of Two-Part MDL.

Two-Part MDL Revisited

As we have seen before, Two-Part MDL encodes an amount of data using a two-part code. One part describes a model, the second part describes the data using this model. The idea of MDL is to make a trade-off between the number of bits that is needed to describe a model and the number of bits that is needed to describe the data using this model. We aim to find relatively simple models that capture much of the regularity that is present in the data. Despite the appeal and naturalness of this idea, there are a few drawbacks. One of these drawbacks stems from the differences between ideal-MDL and applied MDL. We do not use the set of all computable models as model class when applying MDL, but only a small part of it.

The Interval Function Selection Problem

In the problem of finding the right interval function, the model class, $\mathcal{M}$, is restricted to the set of all interval functions with m or fewer intervals, where m is the length of the sample. The $KMDL$ coding scheme uses a smaller model class. The model class is restricted to all *optimal* interval functions for any fixed number of parameters, given some set of x-data. These optimal interval functions use a subset of the x-data as parameter values. In itself this is just using a-priori knowledge on the problem domain to streamline a search process, as we encounter in many fields of Artificial Intelligence and it appears to be a good thing. A consequence of this restriction on the model class is that all models can now be identified by specifying the number of parameters t and the index given to the set of parameters in a lexicographical enumeration of all subsets with t elements of the x-data. So the original model class, $\mathcal{M}$, is transformed into a new model class, $\mathcal{M}_K$, that consists of 2^m models, as a set of m elements has 2^m subsets. Now we look at the number of models in $\mathcal{M}$, the model class used by the MDL_{tp} and MDL_{pss} coding schemes. For each number of bits $p \geq 0$ to code parameters, $\binom{2^p}{t}$ different models for each number of parameters t with $0 \leq t \leq m$ are present in $\mathcal{M}$. As p can take any integer value greater than 0, the size of $\mathcal{M}$ is infinite, though countable and *not* dependent on m.

Our experiments have shown that the performance of Two-Part MDL considerably improved when the MDL_{pss} coding scheme was used instead of the $KMDL$ coding scheme. Now we present a more general discussion of the latter observation.

Discussion

Recall that we use the code length of a model as a measure of the complexity of the model. As we are restricting our model class further, the code lengths of these models will differ more from the code length that they would get when the model classes would not have been restricted. This demonstrates that we are using a measure of (model) complexity that is not only dependent on the model itself but also on the model class. We mean that the code length for a model does not represent a measure for its *individual complexity*. Instead the code length gives us a measure of *complexity relative to the model class*. So by fixing a model class $\mathcal{M}$ and a prefix-free coding scheme $C : \mathcal{M} \to \{0,1\}^*$ we induce a function $L_h : \mathcal{M} \to \mathbb{N}$, such that $L_h(M)$ is the length of $C(M)$.

Recall that Two-Part MDL selects a model by minimizing the evaluation function $L(D) = L_h(M) + L_d(D|M)$ over all models $M \in \mathcal{M}$, where $L_d(D|M)$ denotes the code length of the data D using model M. The complexity term, $L_h(M)$, in the sum prevents us from selecting models that are very complex in terms of $L_h(\cdot)$. Now consider restricting some model class $\mathcal{M}_1$ to $\mathcal{M}_2$, so $\mathcal{M}_2 \subset \mathcal{M}_1$. Suppose that we used coding scheme C_1 and evaluation function L_{h1} with model class $\mathcal{M}_1$, and coding scheme C_2 and evaluation function L_{h2} for

$\mathcal{M}_2$. As $\mathcal{M}_2 \subset \mathcal{M}_1$ we can (in principle) use a coding scheme C_2 with $\mathcal{M}_2$ such that for all $M \in \mathcal{M}_2 : L_{h2}(M) \leq L_{h1}(M)$. Because we have to code fewer models with C_2 the short codewords of C_1 that are not used anymore can be used to code other models in $\mathcal{M}_2$. The error of the models will still be the same if we shrink the model class. Therefore the sum $L_h(M) + L_d(D|M)$ will not get bigger if we use $\mathcal{M}_2$ instead of $\mathcal{M}_1$. So we conclude that if we shrink model class $\mathcal{M}_1$ to $\mathcal{M}_2$, then Two-Part MDL may tend to select more complex models[1] in terms of L_{h1}. This is favorable as long as the models that are selected are not too complex, in the sense that they will not overfit the data.

Now suppose we use information provided by the sample to shrink the model class. Then the idea that a model with small code length is likely to contain little sample specific information and vice versa becomes less evident. We may very well have that even models with short code length contain relatively much sample-specific information. Models that contain relatively much sample-specific information are likely to achieve small error on the sample. Therefore, we expect Two-Part MDL to select models that contain much sample specific information. This may well lead to performance we would call overfitting. This is why we prefer using model classes that are sample-independent. However, we think it is a good idea (see Section 8.1) to use sample-specific information to *select the models which we will actually evaluate*, as long as the computed code length is based on using a sample-independent model class.

8.2.2 Guaranteed Risk Minimization

When we look at GRM, we see that the penalty for complexity is represented in the model evaluation rule (the quantity which we want to minimize) in two different ways, dependent on whether the model class is finite or not. If we take the number of examples, m, available to the model selection method as fixed, it turns out that for finite model classes the complexity term depends on the size of the model class. If infinite model classes are used, the complexity term is dependent on the Vapnik-Chervonenkis dimension of the model class.

We examine the effect of restricting the model class to optimal models once more. The VC dimension of the smallest model class containing some optimal model $\hat{h}_t$ is still t. However, restricting the model class of all interval functions to optimal models for some given set of x-data does imply that we move from an infinite set of models to a finite set of models. Therefore, the penalty term is not any longer dependent on the VC dimension of the model class but merely on its size. We saw that making the model class smaller causes (ideal[2]) Two-Part MDL directly to select more complex hypotheses. Now we examine in which cases shrinking the model class makes GRM assign a lower complexity term to the models in that class. This is the case if:

1. an infinite model class is shrunk to another infinite model class, such that the VC dimension of the new model class is lower than the VC dimension of the original model class. Intuitively this is because if the VC dimension decreases, the model class has apparently become less expressive. Therefore the models in this class are more favorable for modeling the data in the light of the *Occam's Razor*[3] principle[4].

[1] We say M_1 is *more complex in terms of* $L_{h1}(\cdot)$ than M_2 if $L_{h1}(M_1) > L_{h1}(M_2)$.

[2] Applied MDL will attach lower importance to the complexity term only when the codes can actually be made shorter by the code construction algorithm. We might discard some models without being actually able to assign shorter codes to the remaining models, for example because we cannot think of any coding scheme that gives us shorter codes.

[3] See for example [KV97]

[4] Note that the models in this model class are unlikely to contain much sample-specific infor-

2. we transform an infinite model class into a finite model class. In this case more complex models will be preferred if (using Equation 5.2 and 5.5):

$$\ln N < 4VCD(\ln(\frac{2m}{VCD}) + 1) + \ln \delta,$$

where N is the model class size of the finite model class, δ is the confidence parameter and VCD is the VC dimension of the infinite model class.

3. a finite model class is transformed into a smaller finite model class.

In the case where a finite model class is transformed into a smaller finite model class the same danger lurks upon us as with Two-Part MDL: *The complexity term will decrease as the model class becomes smaller, while the models will make the same errors.* This renders especially (some) models in the sub-model-classes (those which we would intuitively call complex) more favorable. This is true because they make a relatively low training error and therefore they benefit more from a decrease in the complexity term. Just as we remarked in the previous subsection on Two-Part MDL, it is good to prefer complex models, just as long as it does not lead to overfitting. We return to this issue in the next section.

8.2.3 Predictive MDL

As we have seen in the chapter on PMDL, this method does not evaluate models by evaluating the sum of an error term and a complexity term. The evaluation of a model class is purely based on the performance (the error made on the sample) of models from that model class. The PMDL method is therefore not sensitive to using some knowledge on the problem domain to shrink the model class. If we do not remove any good models for the data, the shrinking of the model class will only increase the performance of PMDL in terms of shorter running time.

8.3 Overcoding by Using Sample-specific Information

We saw in the second series of experiments (Chapter 7) that Two-Part MDL can overfit when we use sample-independent information on the problem domain, as well as when we use sample-specific information.

For the use of sample-independent information, it is unclear how much knowledge we can use to shrink the model class before this will lead to overfitting.

However, it is easy to see that the use of sample-specific information to code, and thus evaluate, the models involves some risk. Suppose that we have a large parameter space, a small sample and that the examples provide us with good parameter values for parameters v and w. Now suppose that parameter v has far fewer possible values than parameter w. Coding the parameter values using the sample allows us to code the values of both parameters in the same amount of bits. We just have to specify the index of the example that provides us with the parameter value. Thus, modeling using the 'more complex' feature, w, of the data, becomes equally favorable as modeling using a 'simpler' feature, v, of the data. *In general, we prefer simple models over complex ones, if the models perform equally well. Using the data to code models does not represent this preference.*

Another, related, point is that when we use the sample to code the model, the difference between large scale and small scale modeling disappears. Suppose we do

mation because the model class is not very expressive. So if some model makes a small error on the sample, it is likely that the model captures some regularity in the data, where the regularity is not sample-specific.

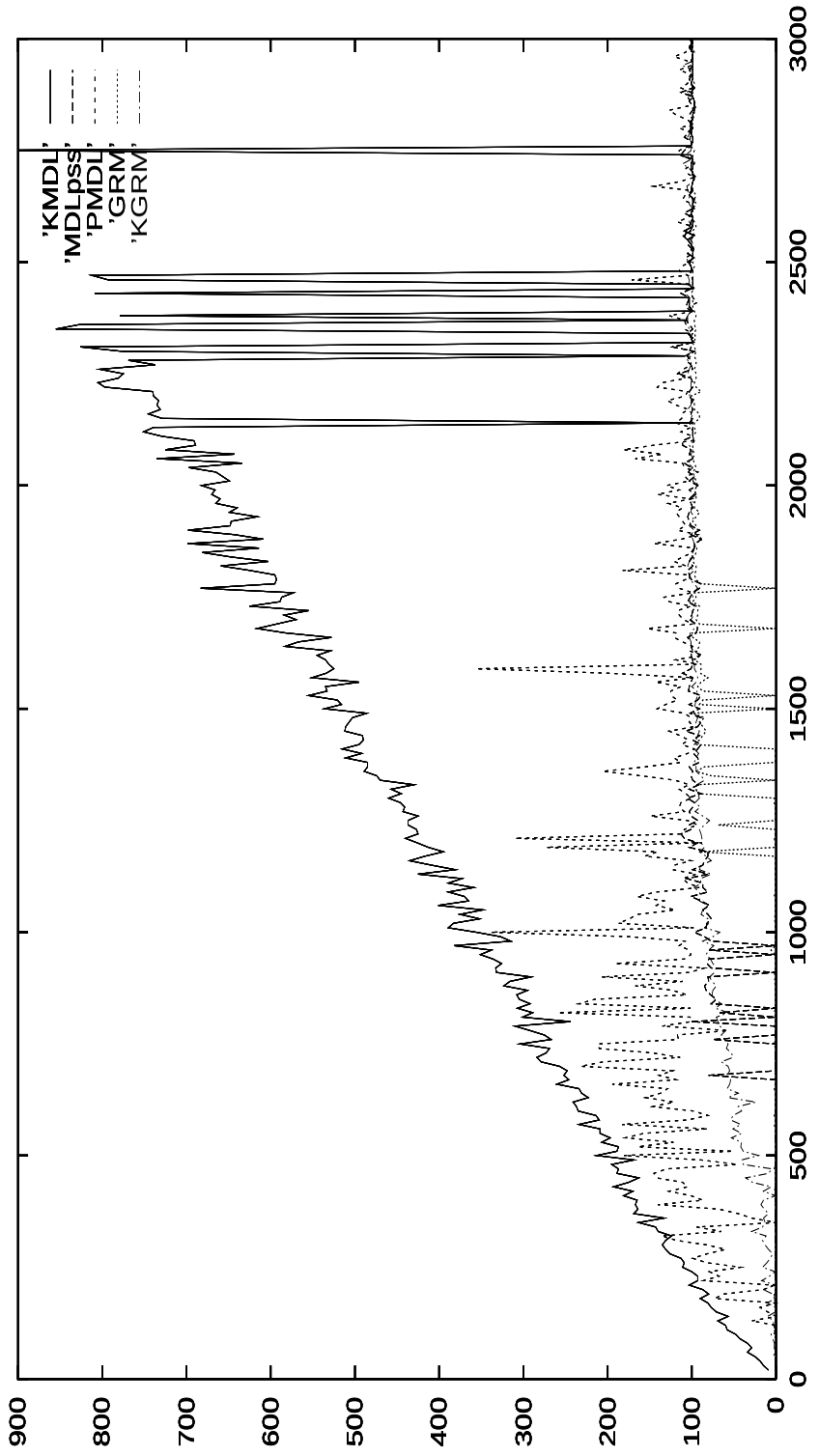

Figure 8.1: Plots of the selected model complexity yielded by: PMDL, $KMDL$, MDL_{pss}, GRM and $KGRM$.

not use the sample to code models and that we want to incorporate some feature in the model that enables us to make a decision based on the values of examples within a very small region of a (numerical) scale. To do so we would need many bits to be able to code values in such a specific region. To model values on a larger scale we would need fewer bits. Thus it would be 'cheaper' to model patterns in the data on a large scale than on a small scale. However, if we use the sample to code the models, we would need the same amount of bits to code small scale patterns as we would to code large scale patterns[5]. We just have to specify which examples in the sample provide us with the correct parameter values. Notice that especially small samples are not very likely to give reliable information on small scale. Small samples may not reveal the small scale patterns. Furthermore, the small scale patterns are more sensitive to noise[6]. The result of all this is that using the sample to code models increases the risk of modeling too much detail and/or incorrectly modeling details.

Finally we discuss a more subtle point. Suppose we use some model, M, that is coded with the use of the data S_1 on the basis of which we selected the model. In general, let S^x denote the x-data of sample S. Let M' denote the description of M given S_1^x. Data sequences S', different from S_1, may be rendered more likely on the basis of their x-data, although the model M is a function; that is: some dependency $X \rightarrow Y$. Now we use M to code some sample S_2. The x-data of S_1 is available, otherwise we could not derive M from M'. We could use S_1^x to code S_2^x. The number of bits $L(S_2^x|S_1^x)$ needed to do so depends (in an ideal setting) on $K(S_2^x|S_1^x)$, the Kolmogorov Prefix Complexity of S_2^x given S_1^x. So data sequences S_2 for which we can describe S_2^x in few bits given S_1^x get very short code words. We could, equivalently, state that the model will assign (too) high probability to S_2. The former phenomenon will also take place if the model makes a large error on the sample S_2.[7] As we wanted to model a dependency $X \rightarrow Y$, *we do not want the models to assign probabilities to samples mainly on the basis of the x-data* of the sample, instead *we want to assign probabilities to data sequences on the basis of how x and y-data are related.*

In the next section, we will summarize what we learned during this project.

8.4 Conclusions

From the two series of experiments and the analysis of these experiments we learned the following:

1. **Overfitting**

   (i) Using information that enables us to code extra model complexity in relatively few bits (as compared to the number of bits needed to code errors of the model) may result in overfitting for small sample sizes. This is true both for modeling indicator functions (Chapter 4) as well as for density estimation (Chapter 7).

   (ii) Using the x-data to code models is one specific way to code parameter values in few bits relative to the bits needed to code the training error. So the use of the x-data to code parameters of models might lead to overfitting. Although we cannot give any theorem that tells that using the data to code models is bad, we saw that there is no real advantage in

[5] We assume that there are values present in the sample to code the small scale patterns.

[6] This is actually dependent on the type of noise present in the data generating function.

[7] Note that the description of the y-data or classification in general takes far fewer bits than the description of the x-data.

using the data to code models, though we do take a higher risk to overfit the data.

(iii) The phenomenon just described does not only affect Two-Part MDL. GRM for finite model classes also performed overfitting. Both methods tend to overfit when using too small model classes.

(iv) It is a hard problem to design a general learning algorithm that does not overfit, though it became clear that it is wise to keep the model class sample-independent. However, how small the model classes should get before overfitting takes place is unclear. In more general terms: there is no optimal way to tune the penalty term of penalty based algorithms, as shown in [KMNR97] and [Wol93].

2. **Good Performance for a Non-Parametric Method**

   We saw that the Predictive MDL method yielded a good performance, although it did not perform much better than Cross Validation using the best value for γ. PMDL has the advantage that we do not have to find this parameter value. PMDL did perform some overfitting. However, it converged at about the same speed as MDL_{pss} to the right model complexity. In terms of generalization error, the performance of PMDL was approximately the same as that of $KGRM$. Note that $KGRM$ was tailored for the specific problem domain, whereas PMDL could be applied in the general way for indicator functions.

3. **Parameter Estimation**

   The first experiments suggested also that the MDL Two-Part principle performs well on selecting a model class, but not on selecting a model within a certain model class. At least this turns out to be the case for small sample sizes, which is the typical situation when performing inductive inference.

Appendix A

Issues of Implementation

In this appendix we treat the questions how we can find optimal models for a given set of data in the interval function selection problem domain.

In the last section of this appendix (page 88) we also give the algorithm that was implemented to obtain the experimental results discussed in Chapter 7.

A.1 Constructing Hypotheses: Introduction

In this section we will discuss some matters of implementation of the algorithms that were used to evaluate the various model selection methods.

When we use the MDL_{tp} or MDL_{pss} coding scheme we must code our parameters with finite precision. The question is how many bits precision would lead to the shortest description of the data? In general, there is no way to compute the optimal precision in advance. So we need to check exhaustively which precision yields the shortest code for a given sample.[1]

The problem has now become an optimization problem in two dimensions. The first dimension concerns the number of parameters (switches) of the model. The second dimension concerns the number of bits used to code each parameter. Now we introduce the concept of *optimal hypothesis*:

Definition A.1 *An* **optimal hypothesis** *(relative to some sample S) denoted by $\hat{h}_{t,p}$, is a hypothesis with t parameters, each of which are encoded using p bits, that has the lowest training error $\hat{\epsilon}(h_{t,p}, S)$ of all hypotheses with t parameters encoded using p bits each.*

Now imagine a two dimensional table in which we have optimal hypotheses for each combination of p (number of bits used to code parameters) and t (number of parameters).

In the next two subsections we present a method to determine the optimal hypothesis for certain values of t and p. We start by giving an algorithm to find the best optimal model for the data using a fixed precision to code the parameters. Next, in Section A.3, we discuss how we can transform this model in an optimal model with fewer parameters. Furthermore, it would be useful to have a criterion to be able to discard the hypotheses that will not lead to data compression in an early stage. This matter is treated in Section A.4. Finally we present in Section A.5 the algorithms that were implemented for the first series of experiments (see Chapter 4, 5 and 6).

[1] Under certain assumptions on the data generating system it suffices to take $\frac{1}{2}\log n$ bits precision when modeling n data items. See [Grü98] for details.

A.2 Finding an Optimal Hypothesis

We start with another definition. Informally, a labeling of length m is the output of a hypothesis or model if it is given some sequence of m x-values as input.

Definition A.2 *A* **labeling** *(or* **classification***) in a classification domain Y of a sequence x^m of m example is a sequence*

$$l^m = (l_1, l_2, ..., l_m), \forall_i (1 \leq i \leq m) \Rightarrow (l_i \in Y).$$

How can we create an optimal hypothesis for the sample given a finite precision p? When we are allowed to encode our parameters using only p bits, we have 2^p different positions where we may define a switch for the hypothesis. The question is then how many switches should be used and where they should be put them. One can also look at this problem as having 2^p intervals that have to be labeled optimally with respect to the data. They have to be labeled such that the lowest possible training error is achieved.

We create an initial hypothesis by labeling the data-sample of length m with lowest possible error, when restricting parameter values to $k2^{-p}$, where k is a non-negative integer between 1 and 2^p.

The idea of the following algorithm is to put a switch between two examples that are labeled oppositely. If this is not possible, *no* value between the examples is codable in p bits. In this case we determine whether there are more positive or negative examples in the interval that is bound by those values that *are* codable in p bits and are closest to the two examples. We label the examples in this interval accordingly. The algorithm **GENERATE** is given by:

INPUT:

- Sample $S(m) = ((x_1, y_1), ..., (x_m, y_m))$
- Number of bits for precision p.

OUTPUT:

- Labeling l^m of data.
- Optimal hypothesis $\hat{h}_{c,p}$ with c switches encoded in p bits per switch.

ALGORITHM:

1. Label the first example as in the data-sample: $\hat{h}_{c,p}(x_1) = y_1$.
2. i=1.
3. **While** $i < m$
   (1) **If** $y_i \neq y_{i+1}$ **and** $\lceil x_i \cdot 2^p \rceil = \lceil x_{i+1} \cdot 2^p \rceil$ **Then**
      1. $min = \arg\min_j \{\lceil x_j \cdot 2^p \rceil = \lceil x_i \cdot 2^p \rceil\}$
      2. $max = \arg\max_j \{\lceil x_j \cdot 2^p \rceil = \lceil x_i \cdot 2^p \rceil\}$
      3. $weight = |\{x_j : y_j = 1, min \leq j \leq max\}| - |\{x_j : y_j = 0, min \leq j \leq max\}|$
      4. **If** $weight > 0$ **Then** $label = 1$
      5. **If** $weight < 0$ **Then** $label = 0$
      6. **If** $weight = 0$ **Then** $label = \hat{h}_{c,p}(x_{min-1})$
      7. $\forall_j (min \leq j \leq max) \Rightarrow (\hat{h}_{c,p}(x_j) = label)$
      8. $i = max$

   (2) **Else** $\hat{h}_{c,p}(x_{i+1}) = y_{i+1}$
4. $i = i + 1$

A.3 Deriving a Simple Hypothesis from a Complex One

We can look at a hypothesis $\hat{h}_{d-1,p}$ as an optimal partition of $[0,1]$ into d intervals with respect to the sample $S(m)$. Now we transform this partition into a partition in which one of the original intervals is deleted. Before we can give the algorithm, we should first define the advantage of an interval.

Definition A.3 *The* **advantage** *of an interval I with respect to sample S and hypothesis h, denoted as $adv(I,S,h)$, is defined as the number of examples in S that is labeled correctly by the hypothesis, h, on the interval I, minus the number of examples in S that is labeled incorrectly by h on the interval I:*

$$adv(I,S,h) = \\ |\{x_k : h(x_k) = y_k, x_k \in I \cap S\}| - |\{x_k : h(x_k) \neq y_k, x_k \in I \cap S\}|$$

Often $adv(I,S,h)$ is referred to by $adv(I)$ when S and h are clear from the context.

Now the algorithm **TRANSFORM** can be stated as follows:

INPUT:

- Sample $S(m)$.
- Hypothesis $\hat{h}_{d,p}$.

OUTPUT:

- Hypothesis $\hat{h}_{d-2,p}$.

ALGORITHM:

1. Compute for each interval $I_i (1 < i < d)$ its advantage $adv(I_i, S, h_{d,p})$.
2. Compute the sum of $adv(I_1)$ and $adv(I_d)$.
3. Delete the interval which has the minimum advantage. Interval I_1 and I_d are regarded as one interval here because we want to create a new hypothesis with two switches fewer than the original.

Lemma A.1 *Let $S(m)$ be a sample and $d,p > 0$ be numbers. If $h_{d,p}$ is optimal for $S(m)$ and $h_{d-2,p}$ is obtained from $h_{d,p}$ using algorithm* **TRANSFORM**, *then $h_{d-2,p}$ is optimal for $S(m)$.*

Proof For clarity we will omit the subscript p. Suppose, by way of contradiction, that h_{d-2} is not optimal. Then there is an optimal hypothesis g_{d-2} with $d-2$ switches, such that $\hat{\epsilon}(g_{d-2}) < \hat{\epsilon}(h_{d-2})$. Let $s_1, \ldots, s_{d-2}$ and $t_1, \ldots, t_d$ be the positions of the switches for g_{d-2} and h_d, respectively. Let $I_0, \ldots, I_d$ be the corresponding intervals for h_d: $I_0 = [0, t_1)$, $I_1 = [t_1, t_2)$, etc.

We assume any hypothesis labels its first interval with a 0.

Hypothesis h_{d-2} has been obtained from h_d by deleting the switches around the interval I_j (possibly $I_j = I_0 + I_d$) that had minimal advantage $adv(I_j)$. We will show that there exists some I_k, $1 \leq k \leq d-1$, which is labeled oppositely by g_{d-2} and h_d. Suppose this is not the case. Then we must have $s_1 < t_2$, for otherwise I_1 would be labeled oppositely, see Figure A.1. Moreover, it is easy to see that whenever $s_i < t_{i+1}$, we must have $s_{i+2} < t_{i+3}$, otherwise I_{i+2} would be labeled oppositely. It now follows inductively that $s_{d-2} < t_{d-1}$. But then I_{d-1} is labeled oppositely. Hence there must exist some I_k labeled oppositely by g_{d-2} and h_d.

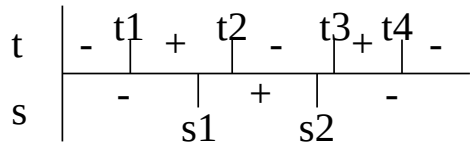

Figure A.1: If both hypotheses start with negative labeling, then there must be some interval $I_k, 1 \leq k \leq d-1$ that is labeled oppositely.

Define g_d by adding switches t_k and t_{k+1} to g_{d-2}. Now, g_d is like g_{d-2} except that it now labels I_k the same as h_d, so g_d makes $adv(I_k) \geq adv(I_j)$ fewer errors than g_{d-2}. Now $\hat{\epsilon}(g_d) = \hat{\epsilon}(g_{d-2}) - adv(I_k)/m < \hat{\epsilon}(h_{d-2}) - adv(I_j)/m = \hat{\epsilon}(h_d)$, contradicting the optimality of h_d.

Suppose that h_d and g_{d-2} label their first interval oppositely, then a similar argument can be given and this argument is left to the reader. Though in this case it is also possible that both I_0 and I_d are both labeled oppositely instead of some $I_k, 1 \leq k \leq d-1$. □

Lemma A.1 enables us to find half of the optimal hypotheses in the table which we introduced at the start of this section, by using the two algorithms just presented. The problem is that we can only transform an optimal hypothesis in one with two switches fewer. To get the other half of the hypotheses we need to derive the optimal hypothesis $h_{c-1,p}$ from the hypothesis $h_{c,p}$ which yields the lowest possible training error among all hypotheses that have parameters encoded in p bits each. Fortunately this is fairly easy. We just delete from $h_{c,p}$ the first or the last interval, depending on which has the lowest advantage. The latter method can be explained as follows: First, note that we only want to model switches that occurred in the data, so we only want to use switches that occur in $\hat{h}_{c,p}$[2]. As a consequence $\hat{h}_{c-1,p}$ will label either the first or the last interval oppositely as compared to $\hat{h}_{c,p}$. Suppose we remove some switch i with $1 < i < c$. If we label the first interval of $\hat{h}_{c-1,p}$ according to $\hat{h}_{c,p}$, then intervals $I_0, \ldots, I_{i-1}$ are labeled according to $\hat{h}_{c,p}$ and intervals $I_i, \ldots, I_c$ are labeled oppositely as compared to $\hat{h}_{c,p}$. Thus it would be better to label just I_c oppositely to $\hat{h}_{c,p}$. A similar argument can be given if we label the last interval of $\hat{h}_{c-1,p}$ according to $\hat{h}_{c,p}$.

A.4 Discarding Useless Hypotheses at an Early Stage

Although we can find all hypotheses to fill the table with optimal hypotheses, there are some entries in the table for which we do not even need to search for the optimal hypothesis.

We can state a few things about the precision p in relation to the number of parameters d and the error $\hat{\epsilon}(\hat{h}_{d,p})$.

- The number of switches d should not exceed $2^p - 1$, because it would otherwise not be possible to assign a different position to each switch. The **GENERATE** algorithm described above makes implicit use of this constraint.

[2]The **GENERATE** algorithm puts as many switches as can be used to improve performance of $\hat{h}_{c,p}$. Thus if we put a switch in some interval in $\hat{h}_{c,p}$ it does not properly represent a switch in the data. Therefore, even if there are examples that are labeled oppositely in some I_i in $\hat{h}_{c,p}$ we do not want to put a switch within I_i. The used precision just does not enable us to represent a switch in the data in these situations.

- We should not consider hypotheses $\hat{h}_{d,p}$ with

$$p > p_{max} = \arg\min_{p}\{\lceil x_i \cdot 2^p \rceil \neq \lceil x_{i+1} \cdot 2^p \rceil : 1 \leq i < m, l(x_i) \neq l(x_{i+1})\}$$

because with p_{max} bits precision we can construct a consistent hypothesis. As we cannot improve the consistent hypothesis with respect to training error, it makes no sense to use more bits or parameters. This only makes the code longer without improving performance.

A.5 Implemented Algorithm

In this subsection we present the various algorithms that were implemented for the first series of experiments. For the first series of experiments we used roughly the same algorithm for each model selection method:

INPUT:

- Minimum number of examples used, min.
- Maximum number of examples used, max.
- Step size between two experiments in examples, $step$.
- Number of switches in target function, d.
- Noise rate of the data source, η.
- Number of runs per setting, n.

OUTPUT:

- For each number of examples that was used output the number of switches that yielded the best evaluation value of a model.

ALGORITHM:

1. $m = min$
2. **WHILE** $m \leq max$
   (1) **FOR** $run = 1$ **TO** n
      1. draw m examples from source with d switches and noise rate η
      2. **FOR** $p = 1$ **TO** p_{max}
         1. find optimal hypothesis $h_{c,p}$ with **GENERATE**
         2. find sub-optimal hypothesis $h_{c-1,p}$
         3. derive all hypotheses with fewer switches with **TRANSFORM**
         4. evaluate each hypothesis
         5. record the number of switches that yielded best hypothesis, where best is defined as having minimal description length using some coding scheme.

   (2) compute average of number of switches that yielded best hypothesis for each number of examples.
   (3) m=m+step

The loop over precision is only executed for the MDL_{tp} and MDL_{pss} model evaluation rule. For Predictive MDL, which we discuss in Chapter 6, a slightly more complicated algorithm was implemented, but the structure should be clear from the algorithm just sketched and the introduction to PMDL in Chapter 6.

A.6 Implemented Algorithm for Clustering Domain

In this section we present the algorithm as it was implemented to obtain the experimental results that are shown in Figure 7.2 in Chapter 7.

INPUT:

- Number of runs, n.
- Minimum number of examples, min.
- Maximum number of examples, max.
- Increase in number of examples between two consecutive trials, $step$.
- Number of normal distributions in data generating function, $clusts$.
- Maximal number of normal distributions in model, c_{max}.
- Maximal number of bits used to encode parameters, p_{max}.

OUTPUT:

- For each model class and for each number of examples that was used, output the mean number of normal distributions and number of bits that yielded the shortest description of the data.

ALGORITHM:

1. **FOR** $run = 1$ **TO** n

   (1) **FOR** $m = min$ **TO** max **STEP** $step$

   1. Generate sample $S(m)$ from mixture of $clusts$ normal distributions, where the mixture is defined as in Equation 7.3 on page 67.
   2. **FOR** $c = 1$ **TO** c_{max}
      1. Let **FORGY** compute clustering C_c on $S(m)$
      2. Estimate on each cluster a normal distribution. We estimate a normal distribution N_i on a cluster C_{t_i} by estimating the ML normal distribution on the data that is assigned to the cluster (the data for which the closest cluster center is that of C_{t_i}).
      3. **FOR** $p = 1$ **TO** p_{max}
         - Truncate precision to p bits for parameters that have to be encoded in model description.
         - Compute for each model class $L(S(m)|C_c)$
         - Compute for each model class $L(C_c)$
         - Check for each model class whether or not the current model yields a shorter code length than previous models; if so, record code length and values of c and p

2. Compute for each model class means for best number of clusters and number of bits for current value of m.

Appendix B

Justification of Error Estimation

We show that it is reasonable to assume the mean case of the noise distribution when making an estimation of the training error (see page 32).

Let F denote the number of examples that are labeled incorrectly by the hypothesis that we are evaluating. Let $\hat{h}_t$ be an optimal hypothesis for some specific sample $S(m)$ with t switches and d the number of switches in the target function. The expected training error is given by:

$$E(\hat{\epsilon}(h_t, S(m)) = E(F/m) = \frac{E(F)}{m} = \frac{t\eta m}{dm} + \frac{1}{2}(1 - t/d) = \frac{d - t + 2t\eta}{2d}$$

See Section 4.2 for an explanation of (B).

Now let's take a look at the variance of the error. When t and d are fixed, the error is only dependent on the actual noise in the correctly modeled area of the [0,1] interval. We saw on page 32 and 33 that with probability $\frac{t\eta}{d} + \frac{1}{2}(1 - t/d)$, an example will be labeled incorrectly by h_t, if we draw the example according to the uniform distribution. The variance of F is $V(F) = m(\frac{t\eta}{d} + \frac{1}{2}(1 - t/d)(1 - \frac{t\eta}{d} + \frac{1}{2}(1 - t/d))$. Now we compare the standard deviation $\sigma(F) = \sqrt{V(F)}$ of F to the expected value of F, $E(F)$. The fraction $\sigma(F)/E(F)$ gives an impression of how great the deviation in F can be compared to the expected value of F. Figure B.1 gives a plot of this fraction for different values of m and t. The plot shows us that already for small sample sizes the deviation of F is relatively small and the assumption of the mean case in the error estimation is justified.

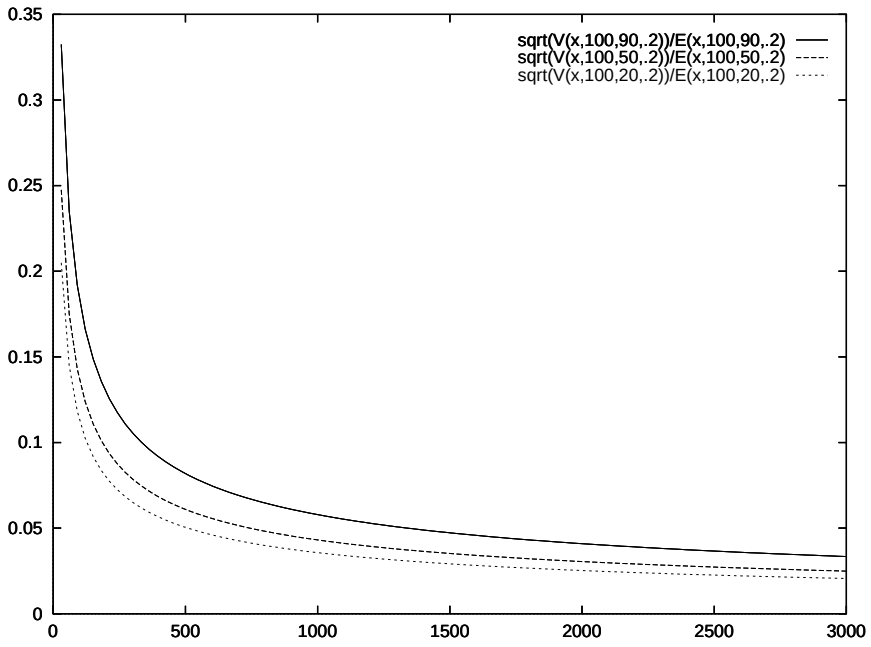

Figure B.1: Plot of the fraction $\sigma(F)/E(F)$ against sample size m. Noise rate $\eta = 0.2$ was used. The value of t was taken 20 (lowest), 50 (middle) and 90 (highest), d was in all cases fixed at 100.

Appendix C

List of Symbols

$\arg\max_x f(x)$:	value of x for which $f(x)$ is maximized.
$\arg\min_x f(x)$:	value of x for which $f(x)$ is minimized.
$\mathcal{C}$:	clustering.
C:	coding scheme.
D:	data sequence.
d_0:	number of parameters needed for consistent hypothesis.
$F_{\mathcal{M}}(\cdot)$:	number of incorrect predictions made by PMDL using model class $\mathcal{M}$.
$f(\cdot)$: density function.	
H:	hypothesis.
$\mathcal{H}(\cdot)$:	entropy.
H_{mld}:	hypothesis that minimizes Two-Part MDL code length for some specific sample.
$\hat{h}$:	optimal hypothesis for some specific sample.
$h_{t,p}$:	hypothesis with t parameters, each encoded in p bits.
$K(\cdot)$:	Kolmogorov prefix complexity.
$I(\cdot\|\cdot)$:	stochastic complexity.
$L^*(\cdot)$:	length of encoding of integer by applying the trick described on page 19 once.
$L_C(x)$:	length of the encoding of x with coding scheme C.
l^m:	labeling.
log:	logarithm to base 2.
ln:	logarithm to base e.
$\mathcal{M}$:	model class, set of models.
M:	model.
$\hat{M}$:	maximum likelihood model.
$\mathbf{m}(\cdot)$:	universal distribution.
$P(\cdot)$:	probability distribution, probabilistic model.
$P_{SC}(\cdot)$:	stochastic complexity distribution.
$\hat{P}(\cdot)$:	maximum likelihood probability distribution.
$S(m)$:	sample of length m.
S^x:	x-data of sample S.
$VCD(\cdot)$:	Vapnik-Chervonenkis dimension.
X^*:	example space.
$\|\cdot\|$:	number of bits needed to denote object in standard binary notation.
$\epsilon(\cdot)$:	generalization error.
$\hat{\epsilon}(\cdot,\cdot)$:	training error.
η:	noise rate.
μ:	mean.
σ^2:	variance.
$\xi(\cdot)$:	estimation of training error.

Bibliography

[AB92] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge University Press, 1992.

[BC91] A. Barron and T. Cover. Minimum complexity density estimation. In *IEEE Transactions of Information Theory*, volume IT-37, 1991.

[BD89] E. Backer and R. Duin. *Statistische patroonherkenning.* Delftse Uitgevers Maatschappij, 1989.

[BRY97] A. Barron, J. Rissanen, and B. Yu. The Minimum Description Length Principle in Coding and Modeling. Forthcoming, 1997.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley, 1991.

[Daw84] A. Dawid. Present position and potential developments: Some personal views, statistical theory, the prequential approach. *Royal Statistical Society*, 147:278–292, 1984.

[GKP89] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics, a Foundation for Computer Science.* Addison Wesley, 1989.

[Grü98] P. Grünwald. *The Minimum Description Length Principle and Reasoning under Uncertainty.* PhD thesis, Centrum voor Wiskunde en Informatica, Universiteit van Amsterdam, 1998.

[Grü99] P. Grünwald. Model Selection based on Minimum Description Length. *Journal of Mathematical Psychology*, page To appear in 1999, 1999.

[KMNR97] M. Kearns, Y. Mansour, A. Ng, and D. Ron. An Experimental and Theoretical Comparison of Model Selection Methods. *Machine Learning*, 27:7–50, 1997.

[Kol65] A. Kolmogorov. Three approaches to the quantitative definition of information. *Problems of Information Transmission*, 1:4–7, 1965.

[KV97] M. Kearns and U. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1997.

[LV97] M. Li and P. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer Verlag, second edition, 1997.

[Ris87] J. Rissanen. Stochastic complexity. *Journal of the Royal Statistical Society*, 49:223–239, 1987. Series: B.

[Ris89] J. Rissanen. *Stochastic Complexity in Statistical Inquiry.* World Scientific, 1989.

[Ris96] J. Rissanen. Fisher information and stochastic complexity. In *IEEE Transactions of Information Theory*, volume IT-42, pages 40–47, 1996.

[Sol64] R. Solomonoff. A formal theory of inductive inference. *Information and Control*, 7, 1964.

[Vap82] V. Vapnik. *Estimation of Dependences Based on Empirical Data.* Springer Verlag, 1982.

[Vap95] V. Vapnik. *The Nature of Statistical Learning Theory.* Springer Verlag, 1995.

[Wol93] D. Wolpert. On overfitting as bias. Technical report, The Santa Fe Institute, 1993.

Index